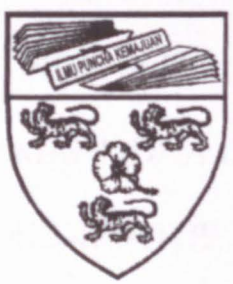
MIKROPEMPROSES 32-BIT
(UNIT KAWALAN)

OLEH:

MOHD. IZUAN BIN MD. YUSOP
WEK000443



FAKULTI SAINS KOMPUTER
DAN
TEKNOLOGI MAKLUMAT
UNIVERSITI MALAYA
KUALA LUMPUR
SESSI 02/03

MOHD IZUAN MD YUSOP

WEK 000443

MIKROPEMPROSES 32-BIT

WXES 3182

PENYELIA:

ENCIK YAMANI IDNA BIN IDRIS

MODERATOR:

ENCIK ZAIDI BIN RAZAK

Penghargaan

Pertama sekali jutaan terima kasih diucapkan kepada penyelia latihan ilmiah I dan II, Encik Yamani Idna bin Idris di atas sokongan, dorongan dan tunjuk ajar yang telah diberikan untuk menghasilkan kertas cadangan dan proses pembangunan sistem bagi projek pembangunan *Mikropemproses 32-bit* ini. Ribuan terima kasih juga diucapkan kepada moderator yang bertanggungjawab di atas projek latihan ilmiah I, Encik Zaidi bin Razak atas nasihat, pandangan, cadangan dan beberapa idea yang diberikan untuk menjadikan sistem ini mantap dan berkesan ke atas pembangunan Mikropemproses 32-bit ini.

Buat keluarga tersayang, emak dan ayah terima kasih atas segala tanggungjawab, keprihatinan dan kata-kata semangat yang diberikan sepanjang menjalankan projek ini. Tidak ketinggalan kepada rakan seperjuangan Suhaimi bin Ishak yang bersama-sama bertungkus-lumus dalam memastikan kejayaan projek ini. Juga tidak lupa ucapan terima kasih kepada rakan-rakan sekuliah atas sokongan yang telah anda berikan dan juga bantuan dalam menghasilkan sistem ini dengan berkesan. Dan akhir sekali kepada mereka yang terlibat dengan secara langsung ataupun tidak langsung dalam pembangunan projek ini, terima kasih di atas segala sumbangan yang kalian berikan.

Abstrak

Mikropemproses merupakan sebuah komponen yang penting di dalam sesebuah komputer. Fungsi bagi sesebuah mikropemproses adalah bergantung kepada senibina dalaman peranti tersebut. Di dalam sesebuah komputer, pembahagian tugas serta unit-unit operasi akan dibezakan dan ditentukan sendiri oleh mikropemproses. Selain daripada itu, mikropemproses atau unit pemprosesan pusat (CPU) juga merupakan satu komponen yang unik dalam senibina komputer. Keunikan sesebuah komputer dapat dilihat apabila keseluruhan CPU disatukan kepada satu pakej kecil iaitu pakej litar bersepadu (IC). Pakej kecil ini menyebabkan penjimatan dari segi saiz dan melibatkan kos yang murah.

Sebelum ini, sesebuah mikropemproses itu dikatakan sebagai kurang ekonomik. Ini dapat dilihat melalui saiz dan kos yang terlibat di dalam pembangunannya. Akan tetapi disebabkan berlakunya revolusi dalam teknologi rekabentuk sistem komputer digital, apabila saiz yang lebih kecil dan kos yang lebih murah, pastinya memberikan kelebihan kepada perekabentuk untuk mereka sebuah struktur mikropemproses. Fungsi sesebuah mikropemproses bukan sahaja terhad dalam kegunaan umum sesebuah komputer atau sebagai unit pemproses di dalam sesuatu sistem yang istimewa. Aplikasi sesebuah mikropemproses dapat dilihat dengan lebih meluas dalam pelbagai kegunaaan kawalan trafik, kawalan ke atas sistem pencucuhan automobil dan aplikasi perniagaan.

Kandungan

Senarai Jadual dan Rajah

Senarai Jadual

Senarai rajah

BAB 1

PENGENALAN

Bab 01 : Pengenalan

1.1 Pengenalan kepada tajuk

Mikropemproses merupakan unit pemprosesan pusat (CPU) yang disatukan dalam sebuah pakej litar bersepadu (IC). Tidak seperti rekabentuk CPU yang biasa, fungsi bagi sesebuah mikropemproses bergantung kepada senibina dalaman peranti tesebut. CPU juga akan membahagikan dan membezakan tugas serta operasi unit-unit dalam sesebuah komputer. Mikropemproses juga merupakan satu komponen yang unik dalam senibina komputer. Apa yang menyebabkan mikropemproses menjadi unik ialah keseluruhan CPU disatukan kepada satu pakej kecil dan disebabkan itulah ianya lebih kecil dan melibatkan kos yang lebih murah.

Disebabkan kos yang murah dan saiz yang kecil, mikropemproses telah mengakibatkan berlakunya revolusi dalam teknologi rekabentuk sistem komputer digital. Ianya memberikan kelebihan kepada perekabentuk untuk mereka sebuah struktur mikropemproses yang sebelum ini dilihat sebagai kurang ekonomik. Mikropemproses berfungsi sebagai CPU di dalam kegunaan umum sesebuah komputer atau sebagai unit pemproses di dalam sesuatu sistem yang istimewa. Aplikasi sesebuah mikropemproses juga boleh didapati dengan lebih meluas seperti kegunaan komputer peribadi di rumah, kawalan trafik, kawalan ke atas sistem pencucuhan automobil dan aplikasi perniagaan.

Selain daripada pembangunan rekabentuk unit kawalan, unit cabang, rekabentuk *program counter* juga akan dibangunkan. *Program counter* seperti yang diketahui berfungsi untuk mendapatkan arahan seterusnya yang akan dilaksanakan daripada ingatan. Rekabentuk yang terakhir ialah rekabentuk unit cabang atau *branch unit*. Sementara unit cabang pula akan dibahagikan kepada dua iaitu *'conditional'* dan

'unconditional'. Kedua-dua keadaan ini akan berlaku pada status yang tertentu semasa perlaksanaan arahan.

1.2 Definisi masalah

1.2.1 Pembahagian tugas

Di dalam membangunkan projek Mikropemproses 32-bit ini, pembahagian tugas telah dilakukan untuk mengelakkan sebarang pertindihan kerja berlaku. Pembahagian tugas ini juga untuk membolehkan setiap seorang memberi tumpuan terhadap komponen-komponen yang cuba dibangunkan.

1.2.2 Menentukan spesifikasi mikropemproses

Spesifikasi di sini bermaksud ciri-ciri prototaip mikropemproses yang akan dibangunkan.

1.2.3 Membangunkan kod sumber (source code)

Setiap komponen yang dibangunkan nanti akan diuji dengan menggunakan VHDL *(VHSIC hardware description language)*. Oleh itu kod-kod yang akan ditulis nanti mestilah memenuhi kehendak sesuatu komponen supaya boleh digabungkan untuk menjadi satu bahasa yang sempurna.

1.2.4 Mengkoordinasi komponen-komponen

Komponen-komponen mikropemproses yang dibangunkan nanti adalah secara berasingan. Jadi adalah penting bagi penggabungan dilakukan di akhir projek nanti.

1.3 Skop projek

Projek mikropemproses 32-bit ini dilakukan oleh penulis dengan rakannya. Pembahagian tugas telah dilakukan untuk memastikan supaya projek ini berjalan dengan lancar dan dapat disiapkan dengan sepenuhnya dalam masa yang telah ditetapkan. Di dalam kajian kali ini, beberapa modul telah dikenalpasti sebagai asas kepada pembangunan sebuah mikropemproses yang sempurna. Modul-modul tersebut ialah:

- Unit pelaksana (FPU)
- Perwakilan Daftar – (Register)
- Set arahan – (instruction set)
- Program counter
- Unit kawalan – (control unit)
- Unit cabang-branch unit

Berikut disenaraikan pula pembahagian tugas yang dilakukan antara penulis dengan rakannya dalam membangunkan projek ini:

Penulis	Rakan, Suhaimi Ishak
Pembangunan unit kawalan	Pembangunan Unit pelaksana-FPU
Pembangunan Program counter	Pembangunan Set arahan
Pembangunan unit cabang-branch unit	Pembangunan daftar

Jadual 1.1 : Jadual Pembahagian tugas

Tumpuan dalam kajian kali ini akan diberikan kepada pembangunan *"program counter"*, unit kawalan dan unit cabang (branch unit). *"Program counter"* merupakan satu daftar khas. Fungsi utamanya dalam sesebuah mikropemproses ialah

untuk mendapatkan arahan seterusnya daripada unit ingatan. Di mana *"program counter"* mengandungi alamat arahan seterusnya yang akan diambil daripada ingatan.

Unit kawalan atau CU pula merupakan antara komponen major dalam sesebuah mikropemproses (CPU). Unit kawalan(CU) akan sentiasa mengawal pergerakan data dan arahan keluar dan masuk ke CPU. CU juga mengawal operasi yang akan dilakukan dalam ALU.

Selain daripada itu, terdapat dua teknik yang digunakan untuk membangunkan satu unit kawalan iaitu implementasi *hardwired* dan *microprogrammed.* Bagi mikropemproses yang akan dibangunkan, unit kawalan *hardwired* digunakan kerana mikropemproses yang akan dihasilkan adalah jenis RISC*(reduce instructions set of computers)*

Unit cabang atau dalam istilah bahasa inggerisnya *branch unit* terbahagi kepada dua iaitu *'conditional'* dan *'unconditional'*. Biasanya cabang *'unconditional'* akan melompat(jumps) kepada baru, iaitu luar daripada aliran arahan. Cabang *'conditional'* pula akan menguji operasi sebelumnya sama ada perlu ataupun tidak untuk unit cabang bertindak. Sebagai contoh, unit cabang hanya akan di ambil jika keputusan sebelumnya adalah bernilai *negative*. Data yang telah diuji untuk cabang *'conditional'* akan disimpan di dalam lokasi istimewa di dalam CPU yang dipanggil bendera(flags).

1.4 Objektif/Tujuan

Di dalam menghasilkan mikropemproses ini, beberapa objektif telah dikenalpasti dalam membangunkannya. Antara objektif-objektif tersebut ialah:

- Objektif utama dalam kajian ini ialah untuk menghasilkan sebuah prototaip mikropemproses 32-bit yang mampu berfungsi dengan baik.

- Menghasilkan komponen unit kawalan dan *"program counter"* yang dapat berfungsi dengan menggunakan VHDL(VHSIC Hardware Description Language). VHSIC pula adalah singkatan kepada *Very High Speed Integrated Circuits.*
- Memahirkan diri dengan penggunaan VHDL

1.5 Kekangan

Jika dilihat pada masa sekarang, zaman di mana teknologi yang semakin pesat membangun, pastinya pembangunan mikropemproses 32-bit ini nanti amat terhad akan kebolehgunaannya. Fokus yang utama di sini ialah untuk menghasilkan sebuah mikropemproses yang mampu berfungsi atau lebih tepat lagi asas kepada pembentukan sebuah mikropemproses yang mudah. Berikut merupakan kekangan yang dikenalpasti:

- Mikropemproses 32-bit yang dibangunkan ini pastinya tidak dapat dibandingkan dengan mikropemproses yang ada sekarang yang sampai ke tahap tertinggi sebagai contoh Pentium 4.
- Di samping itu, faktor masa juga menjadi kekangan dalam menghasilkan sebuah mikropemproses. Pembangunan sesebuah komponen itu memakan masa yang agak panjang dalam menghasilkannya serta memastikan ianya dapat berfungsi. Disebabkan itulah pembangunan mikropemproses dalam kajian kali ini hanya mempertimbangkan komponen-komponen asas seperti FPU, unit kawalan(CU) dan perwakilan daftar(register).
- Selain itu, unit kawalan yang dihasilkan juga tidak membincangkan tentang bus kawalan yang menjadi perantara bagi setiap komponen di dalam mikropemproses.

- Hanya melaksanakan arahan data berjenis titik apung sahaja dan operasi yang dilakukan terhad kepada operasi tambah, tolak dan darab.
- Arahan-arahan yang dilaksanakan nanti hanyalah berbentuk arahan yang mudah dan mikropemproses ini tidak boleh digunakan sebagai unit pemprosesan pusat bagi sesebuah computer.
- Sebarang kesilapan boleh menyebabkan beberapa komponen tidak dapat berfungsi dengan baik. Namun masalah ini akan cuba diselesaikan.

1.6 Penyelesaian masalah

Bagi setiap masalah yang dinyatakan di atas, beberapa langkah telah di ambil untuk mengelakkan ianya daripada berterusan ataupun sekurang-kurangnya dapat mengurangkan masalah ke tahap yang minimum. Antaranya ialah:

- Pertemuan yang lebih kerap antara penulis dengan rakan sekerja bagi membincangkan masalah yang dihadapi.
- Berjumpa dengan penyelia bagi membincangkan masalah yang dihadapi disamping meminta pendapat terhadap projek yang sedang dijalankan.
- Rujukan ke atas sampel-sampel kod VHDL yang berkaitan dengan projek yang dijalankan

1.7 Penjadualan

Aktiviti Pembangunan	Jun '02	Julai '02	Ogos '02	Sept '02	Okt '02	Nov '02	Dis '02	Jan '03	Feb '03
Kajian Sistem dan Rujukan	←	→							
Analisis Keperluan		←	→						
Rekabentuk SIstem			←	→	→	→	→	→	
Pengkodan				←	→	→	→	→	
Pengujian					←	→	→	→	
Implementasian							←	→	
Dokumentasi			←	→	→	→	→	→	→

Jadual 1.2 : Penjadualan Projek

BAB 2

KAJIAN LITERASI

Di dalam pembangunan mikropemproses 32-bit ini, terutamanya terhadap unit kawalan dan juga *program counter*, beberapa kajian telah dilakukan terhadap mikropemproses yang sedia ada. Kajian dilakukan terutamanya terhadap pemproses RISC(reduce instructions set computers). Ini adalah kerana mikropemproses yang akan dibangunkan nanti adalah jenis RISC.

2.1 Mikropemproses

Mikropemproses merupakan perkara asas dan merupakan komponen terpenting dalam penghasilan komputer peribadi. Tanpa mikropemproses boleh dikatakan komputer hanyalah seumpama mesin taip. Akan tetapi aplikasi mikropemproses bukan sahaja terhad kepada penggunaan dalam komputer sahaja, tetapi turut melibatkan peranti dan mesin automasi. Aplikasi mikropemproses merangkumi daripada kegunaan domestik seperti mesin basuh, ketuhar gelombang mikro, televisyen, automobil dan lain-lain sehinggalah ke industri berat seperti hidraulik dan lain-lain lagi.

Pada dasarnya, komputer mempunyai unit pemprosesan pusat(CPU), peranti input/output dan ingatan. Tradisinya CPU dibahagikan kepada dua bahagian iaitu:- unit logik aritmetik atau dalam istilah inggeris *arithmetic logic unit(ALU)* dan unit kawalan atau dalam istilah inggeris *control unit(CU)*. Kedua-dua ALU dan CU merupakan komponen yang diskrit di atas papan yang berlainan disebabkan faktor saiz kedua-dua komponen. Apabila semikonduktor diperkenalkan dan pengeluaran lita bersepadu(IC) bermula, komponen-komponen tadi disepadukan dengan drastik ke tahap di mana kedua-duanya dipadatkan menjadi satu cip yang dinamakan mikropemproses.

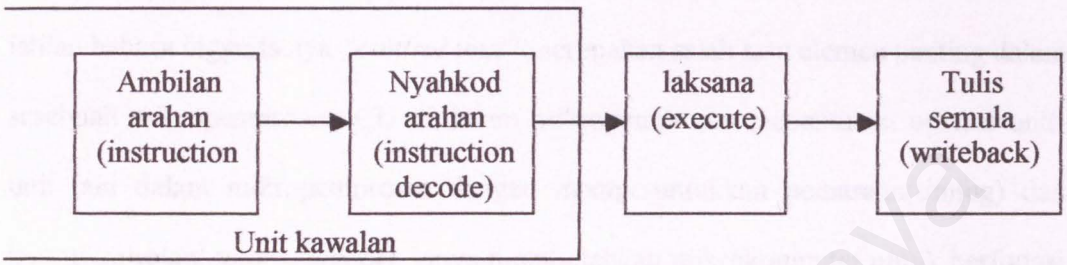
Disebabkan itulah, dalam istilah yang mudah, mikropemproses merupakan CPU dalam LSI atau cip VLSI. Dari segi istilah teknikal pula, mikropemproses merupakan pelbagai guna, boleh diprogramkan, daftar peranti elektronik yang membaca arahan binari daripada peranti storan yang dipanggil ingatan, menerima data binari sebagai input dan memproses data berdasarkan arahan yang diterima dan mengeluarkan keputusan sebagai output.

2.2 RISC

Singkatan bagi *Reduced Instruction Set Computing* dan disebut "risk". Setiap mikropemproses mempunyai satu set arahan dengan mana ia memproses kesemua data. Sehingga pertengahan dekad 80an kebanyakan kemajuan mikropemproses dilakukan dengan menambah jumlah bilangan arahan yang boleh difahami dan dilaksanakan oleh mikropemproses. Tetapi pada ketika itu beberapa pengeluar mikropemproses cuba mencari haluan lain dengan mengeluarkan cip yang hanya boleh melaksanakan beberapa set arahan-arahan yang ringkas sahaja. Terdapat dua kelebihan nyata sistem ini:- cip mikropemproses boleh memproses arahan dengan lebih ringkas dan ini menjadikannya lebih cepat. Disebabkan keringkasan pemprosesan arahan mikropemproses, cip jenis ini tidak memerlukan bilangan transistor yang banyak dan oleh itu kos merekabentuk dan mengeluarkan cip jenis ini adalah lebih rendah.

Konsep bagi pemproses RISC adalah berdasarkan idea di mana asas set arahan bergabung dengan pengkompil yang cekap yang boleh memberikan persembahan lebih baik daripada *Complex Instruction Set Computer* (CISC) dengan jumlah yang besar terhadap arahan tertentu. Dengan memudahkan operasi-operasi tersebut, membenarkan setiap arahan disudahkan dalam satu kitar pemproses.

Asas laluan data bagi pemproses RISC adalah seperti ditunjukkan dalam rajah 2.1 di bawah. Penyahkod arahan akan meletakkan arahan yang ditujukan oleh *program counter* (PC) daripada ingatan pemproses. Penyahkod arahan kemudiannya akan menjana isyarat kawalan yang sesuai untuk unit pelaksanaan, di mana menunjukkan fungsi yang diminta (aritmetik, logik dsb.) ke atas data. Unit tulis semula (*writeback*) akan kemas kini ingatan dengan mana-mana nilai baru.



Rajah 2.1 : laluan data bagi RISC Processor

Kemunculan cip dengan teknologi RISC ini mengakibatkan cip dengan teknologi lama dipanggil CISC atau *Complex Instruction Set Computing*. Namun setelah semua ini diperkatakan, kini tidak terdapat jurang yang terlalu besar di antara teknologi RISC dan CISC. Ini kerana kos pengeluaran cip mikropemproses semakin menurun dan kebanyakan cip CISC telahpun memuatkan arahan-arahan RISC dan banyak cip RISC yang baru mengandungi arahan-arahan CISC yang lama. Jadi nampaknya terdapat semacam "penggabungan" antara keduanya.

2.3 Unit kawalan

Unit kawalan atau CU ini mengandungi litar yang menggunakan isyarat elektrik yang mengarahkan keseluruhan sistem komputer untuk melaksanakan sesuatu arahan. CU tidak melaksanakan apa-apapun arahan sesebuah program sebaliknya mengarahkan bahagian-bahagian tertentu dalam sistem komputer untuk

melakukannya. CU aktif berkomunikasi terutamanya dengan Arithmetik / Logik dan ingatan (memory) bagi memastikan kelancaran sesebuah sistem komputer.

Tajuk utama dalam pembangunan ini ialah mikropemproses 32-bit. Akan tetapi fokus rekabentuk adalah lebih kepada pembangunan unit kawalan, *'program counter"*dan unit cabang*(branch unit)*. Rekabentuk yang akan dibangunkan dalam kajian kali ini ialah berkaitan dengan unit kawalan. Unit kawalan (CU) atau dalam istilah bahasa inggeris nya *"control unit"* merupakan salah satu elemen penting dalam sesebuah mikropemproses. CU di dalam mikropemproses menentukan operasi unit-unit lain dalam mikropemproses dengan memperuntukkan pemasaan(timing) dan isyarat kawalan(*control signal*). Ianya membolehkan mikrokomputer untuk berfungsi bagi melaksanakan program yang disimpan di dalam memori dalam bentuk arahan dan data. CU juga mengandungi logik yang diperlukan untuk mentafsirkan arahan dan menjana isyarat yang diperlukan untuk melaksanakan sebarang arahan.

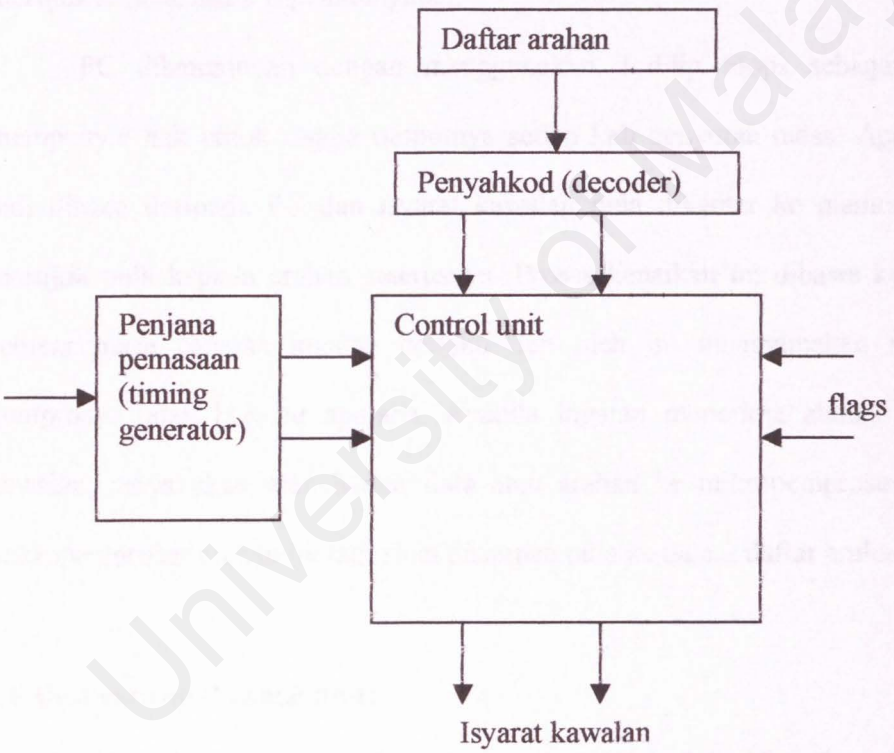
Dua perkataan utama yang digunakan dalam menerangkan tentang unit kawalan ini ialah ambilan(*fetch*) dan laksana(*execute*). CU akan mengambil arahan dengan menghantar alamat dan baca arahan kepada unit ingatan. Arahan yang terdapat pada alamat ingatan akan di pindahkan ke CU untuk penyahkodan. Ianya akan menjana isyarat yang diperlukan untuk melaksanakan arahan.

2.4 Implementasi Unit kawalan

Dua teknik utama yang digunakan dalam mengimplementasikan unit kawalan ialah implementasi *"hardwired"* dan *"microprogrammed"*. Di dalam implementasi *hardwired*, unit kawalan pada dasarnya adalah sejenis litar gabungan. Input isyarat logiknya akan ditukarkan kepada satu set output isyarat logik, atau lebih dikenali sebagai isyarat kawalan (*control signals*). Di dalam implementasi *microprogrammed* pula, ingatan kawalan mengandungi program yang menerangkan tentang sifat unit

kawalan. Menjadi lebih mudah untuk mengimplementasikan unit kawalan dengan hanya melaksanakan program tersebut.

Dalam pembangunan mikropemproses ini, implementasi yang akan digunakan ialah implementasi *hardwired.* Ini disebabkan bagi sebuah RISC(Reduce Instructions Set of Computer) pemproses, teknik yang biasa digunakan ialah teknik implementasi *hardwired.* Teknik ini juga adalah lebih pantas berbanding dengan implementasi *microprogrammed.* Pembangunan mikropemproses ini juga adalah dengan menggunakan kaedah talianpaip(pipelining) untuk mempercepatkan lagi pelaksanaan sesuatu arahan.



Rajah 2.2 : model bagi implementasi hardwired

2.5 Program counter(PC)

Program counter berfungsi untuk mengekalkan trek bagi alamat memori di mana arahan seterusnya yang akan dilaksanakan disimpan. Alamat tersebut akan dihantar ke memori menerusi talian alamat dengan isyarat kawalan yang dihantar oleh talian kawalan. Isyarat ini akan mentafsirkan data yang dihantar daripada memori ke mikropemproses. Terdapat masa terbiar antara penghantaran alamat oleh mikropemproses dan penerimaan data . Pada masa tersebut, pemproses menunggu data dan oleh itu mengurangkan keberkesanannya. Masa ini, disimpan oleh ingatan dalam menghantar data selepas menerima alamar dan isyarat kawalan, tindakan ini merujuk kepada masa capaian ingatan.

PC dibangunkan dengan menggunakan T Flip Flops sebagaimana PC mempunyai hak untuk toggle outputnya setiap kali denyutan masa. Apabila alamat tadi dibaca daripada PC dan isyarat kawalan pula dihantar ke memori, PC akan merujuk pula kepada arahan seterusnya. Proses kenaikan ini dibawa ke seterusnya semasa masa capaian ingatan berlaku dan oleh itu menggunakan masa ketika pemproses tidak berbuat apa-apa. Apabila ingatan menerima alamat dan isyarat kawalan, ianya akan menghantar data atau arahan ke mikropemproses. Di dalam mikropemproses arahan ini tadi akan disimpan pula ke dalam daftar arahan.

2.6 Unit cabang (branch unit)

Unit cabang akan melaksanakan arahan cabang di dalam kawalan program (*program control*). Unit cabang biasanya merupakan satu alamat arahan. Apabila dilaksanakan arahan cabang akan menyebabkan perpindahan nilai alamat kepada *program counter. Program counter* yang mengandungi alamat bagi arahan untuk dilaksanakan, arahan seterusnya akan datang daripada lokasi alamat.

Cabang terbahagi kepada dua iaitu *conditional* dan *unconditional*. Arahan cabang *unconditional* akan menyebabkan cabang akan di tetapkan alamatnya tanpa sebarang keadaan. Arahan cabang *conditional* pula akan menetapkan keadaan di mana cabang akan berfungsi sebagai contoh jika cabang tersebut bernilai positif atau cabang bernilai kosong (zero). Jika keadaan ini berlaku, *program counter* akan bertindak dengan alamat cabang dan arahan seterusnya akan di ambil daripada alamat ini. Jika keadaan ini tidak berlaku, maka *program counter* tidak akan berubah. *Program counter* akan mengambil arahan seterusnya daripada lokasi seterusnya daripada jujukan.

2.7 PowerPC 601

Senibina PowerPC merupakan salah satu pemproses yang menggunakan mesin RISC. Asas bagi 601 adalah terbahagi kepada tiga bahagian unit pelaksanaan talian paip iaitu: integer, titik apung, dan pemprosesan cabang. Ketiga-tiga unit ini boleh melaksanakan tiga arahan dalam satu masa.

2.7.1 Unit hantar (dispatch unit)

Unit hantar akan ambil arahan daripada daripada cache dan memuatkannya kedalam barisan hantar, di mana ianya boleh memegang lapan arahan dalam satu masa. Ianya akan memproses aliran arahan-arahan ini untuk diberikan kepada aliran arahan seterusnya kepada pemprosesan cabang, integer dan unit titik apung. Bahagian atas (*upper half*) bagi barisan bertindak sebagai penimbal untuk memegang arahan sehingga arahan-arahan tersebut dihantar ke bahagian bawah (*lower half*). Ianya bertujuan untuk memastikan supaya unit hantar tidak tertangguh menunggu arahan daripada *cache*. Dalam bahagian bawah, arahan dihantar berdasarkan skema-skema di bawah:

- Unit pemprosesan cabang (Branch Processing Unit) : Mengawal kesemua arahan cabang.
- Unit titik apung (floating point unit) : Mengawal kesemua arahan titik apung.
- Unit integer (Integer unit) : Mengawal arahan integer, menyimpan dan mengeluarkan antara daftar fail dan *cache*, dan juga arahan perbandingan integer

Membenarkan arahan-arahan cabang dan titik apung menjadi isu luar daripada barisan hantar membantu mengekalkan arahan talian paip dalam pemprosesan cabang dan unit titik apung penuh, dan ianya akan menggerakkan arahan melalui barisan hantar bertindak seperti mana biasa.

Unit hantar juga mengandungi logik yang membolehkannya untuk menghitung alamat pra-ambilan (prefetch address). Arahan ambilan akan disambung secara berjujukan sehingga arahan cabang bergerak ke bahagian bawah unit hantar. Apabila unit pemprosesan cabang memproses arahan, ianya akan kemas kini alamat pra-ambilan jadi arahan yang berjaya akan diambil daripada alamat baru dan dimasukkan ke dalam barisan hantar.

2.7.2 Arahan talianpaip (instruction pipelines)

Terdapat kitar ambilan yang lazim bagi kesemua arahan; ianya melibatkan sebelum arahan dihantar ke unit khas. Kitar kedua bermula dengan penghantaran arahan ke unit khas. Ini bertindih dengan aktiviti-aktiviti dalaman unit.

Bagi arahan cabang, kitar kedua termasuklah penyahkodan dan pelaksanaan arahan. Unit integer pula membahagikan dengan arahan yang melibatkan operasi *load/store* dengan memori (termasuklah load/store titik apung), penghantaran daftar-

daftar, atau operasi ALU. Di dalam kes *load/store*, terdapatnya kitar penjanaan alamat diikuti dengan penghantaran keputusan alamat ke *cache*, dan jika perlu ianya akan dihantar kepada kitar tulis-semula. Bagi arahan-arahan lain, *cache* tidak terlibat dan terdapat kitar pelaksanaan diikuti dengan tulis semula kepada daftar.

2.7.3 Pemprosesan cabang (branch processing)

Bagi meningkatkan keupayaan RISC, kunci utama adalah dengan mengoptimumkan penggunaan talian paip. Biasanya, elemen yang paling kritikal dalam rekabentuk bagaimana cabang dikendalikan. Dalam PowerPC, pemprosesan cabang bertanggungjawab terhadap unit cabang. Unit ini direka supaya dalam mana-mana kes, cabang-cabang tidak memberi kesan terhadap kelancaran pelaksanaan dalam unit-unit lain; jenis cabang ini merujuk kepada cabang *zero-cycle*. Untuk mencapai cabang *zero-cycle*, strategi berikut digunakan:

1. Logik dibekalkan untuk mengimbas menerusi penimbal hantar (dispatch buffer) untuk cabang. Target alamat cabang di jana apabila cabang pertama muncul dalam bahagian bawah barisan.
2. Percubaan dilakukan untuk menentukan keluaran bagi cabang *conditional*. Jika keadaan kod telah diset dengan secukupnya, ianya boleh ditentukan. Dalam kes lain, selagi arahan cabang dijumpai, logik ditentukan jika cabang

    - Akan diambil; ini adalah kes bagi cabang *unconditiona*l, dan cabang *unconditiona*l di mana keadaan kod diketahui dan menunjukkan cabang.
    - Tidak akan diambil; ini adalah kes bagi cabang *conditional* di mana keadaan kod diketahui dan menunjukkan tiada cabang.

- Keluaran tidak boleh ditentukan. Dalam kes ini, cabang akan mengambil untuk *backward branches* dan mengagak tidak mengambil *forward branches.* Arahan cabang akan dihantar ke unit pelaksanaan. Sekali keadaan nilai kod dikeluarkan dalam unit pelaksanaan, unit cabang sama ada akan membatalkan arahan dalam talian paip dan meneruskan dengan ambilan target jika cabang diambil, atau isyarat untuk arahan *conditional* dilaksanakan.

BAB 3

METODOLOGI

3.1 HDL (Hardware Description Language)

3.1.1 Pengenalan

HDL merupakan singkatan bagi Hardware Description Language. Ia merupakan suatu bahasa yang digunakan untuk merekabentuk perkakasan.. Dengan menggunakan HDL, kita dapat melakukan operasi simulasi, pengujian, rekabentuk dan dokumentasi bagi menerangkan perkakasan. Bahasa-bahasa ini menyediakan format yang mudah dan padat untuk perwakilan hierarki fungsian dan penyambungan suatu sistem digital. HDL pada asasnya, terdiri daripada suatu set simbol dan notasi yang mudah yang menggantikan gambarajah digital litar skematik. Selain daripada itu HDL secara formal menyatakan dan mungkin mempersembahkan perkakasan dalam satu atau banyak peringkat *abstraction.*

Terdapat dua perisian yang tersedia untuk HDL. Dua perisian ini termasuklah pensimulasi dan program sintesis perkakasan. Bagi *verification* rekabentuk, perisian program sintesis digunakan, manakala pensintesis digunakan untuk penjanaan perkakasan secara automatik. Program penjanaan pengujian mungkin bergantung kepada HDL bagi menyediakan kemudahan dalam bentuk *netlist,* aplikasi pengujian *testbench* dan suntikan ralat.

Salah satu daripada bahasa-bahasa yang digunakan untuk menerangkan rekabentuk sistem elektronik adalah VERILOG (*Verifying Logic*). Tetapi penggunaannya adalah daripada peringkat *Register-Transistor* (RTL) ke bawah.

Selain itu, kebanyakan bahasa yang lain tidak menyokong pendekatan hierarki menyebabkan masalah besar sering timbul apabila merekabentuk tugasan yang kompleks.

3.2 VHDL

3.2.1 Pengenalan

VHDL adalah akronim bagi VHSIC Hardware Description Language (VHSIC pula adalah akronim bagi Very High Speed Integrated Circuits). Ianya merupakan gambaran bahasa perkakasan yang boleh digunakan untuk memodelkan sistem digital. Ianya juga mengandungi elemen-elemen yang boleh digunakan untuk menerangkan tentang struktur sistem digital. Bahasa ini memberikan sokongan terhadap memodelkan sistem secara hierarki dan juga menyokong rekabentuk metodologi atas-bawah dan bawah-atas. Sistem dan subsistem yang lain boleh diterangkan pada mana-mana paras pengabstrakan daripada paras senibina sehinggalah ke paras get.

VHDL adalah suatu standard (VHDL-1076) yang dibangunkan oleh IEEE (*Institute of Electrical and Electronic Engineers*). Bahasa perkakasan ini telah melalui beberapa semakan dan pembaikan. Pada masa kini, versi yang paling banyak digunakan adalah versi 1987 (std 1076-1987). Juga terdapat versi yang baru iaitu versi 1993.

Ia digunakan untuk dokumentasi, *verification* dan sintesis suatu rekabentuk sistem digital yang besar. Ini merupakan ciri-ciri utama VHDL kerana kod VHDL yang sama boleh secara teorinya mencapai ketiga-tiga matlamat tersebut.

Selain digunakan untuk ketiga-tiga tujuan tersebut, VHDL boleh digunakan dalam tiga kaedah yang berbeza iaitu penstrukturan, aliran data dan kelakuan bagi menerangkan perkakasan.

Bahasa VHDL bukan sahaja boleh mentakrifkan sinteks tetapi juga mentakrifkan dengan jelas semantik simulasi bagi setiap bahasa yang dibangunkan.

Disebabkan itu, model-model yang ditulis dalam bahasa ini boleh disahkan dengan menggunakan simulator VHDL.

3.1.1 Kebolehan

Berikut merupakan antara kemampuan yang dapat diberikan oleh bahasa ini yang membezakan ianya daripada lain-lain HDL(*hardware description language*)

- Bahasa ini menyokong hierarki; disebabkan itu, sistem digital boleh dimodelkan sebagai satu set komponen yang saling bersambungan; setiap komponen pula boleh dimodelkan sebagai sambungan bagi subkomponen
- Bahasa ini juga turut menyokong metodologi rekabentuk yang fleksibel: atas-bawah, bawah-atas atau campuran kedua-duanya
- Menyokong model pemasaan *'synchronous and asynchronous'*
- Pelbagai teknik pemodelan digital, seperti mesin *'finite-state'*, algoritmik, dan persamaan Boolean kesemuanya boleh dimodelkan dengan menggunakan bahasa ini.
- Bahasa ini juga terbuka kepada umum, bacaan manusia, bacaan mesin dan kesemua di atas, dan ianya bukan untuk kegunaan persendirian
- Bahasa ini juga adalah dengan standard IEEE dan ANSI; oleh itu, setiap model yang diterangkan dengan bahasa ini adalah bersesuaian
- Bahasa ini juga mengandungi elemen-elemen yang membolehkan rekabentuk model *'large-scale'* dilakukan dengan mudah; contohnya, komponen-komponen, fungsi-fungsi, prosedur, dan pakej
- Model yang dibina bukan sahaja menerangkan tentang kefungsian pada rekabentuk yang dibangunkan, tetapi juga mengandungi maklumat tentang rekabentuk itu sendiri.

- Standard VHDL juga dapat mengubah data rekabentuk dengan lebih mudah daripada pangkalan data rekabentuk.
- Ia turut membenarkan penggunaan banyak senibina dan perhubungan dengan rekabentuk yang sama dalam pelbagai peringkat proses rekabentuk.

BAB 4

REKABENTUK SISTEM

Bab 04 : Rekabentuk Sistem

4.1 Pengenalan

Rekabentuk merupakan peringkat yang paling penting bagi pembangunan projek. Kesemua kesemua komponen-komponen akan diterjemahkan ke dalam ciri-ciri sistem bagi memenuhi fungsi mikropemproses yang akan dibangunkan. Rekabentuk komponen-komponen ini kemudiannya akan disertakan dengan penerangan terhadap fungsi bagi setiap komponen dan interaksi yang wujud antara komponen.

Merekabentuk suatu sistem elektronik bermakna menentukan satu set komponen dan antaramuka komponen yang memenuhi set keperluan yang dinyatakan dalam spesifikasi. Rekabentuk cip melibatkan penstrukturan komponen-komponen atau modul-modul seperti yang telah diterangkan dalam bab 2. Ia merupakan suatu proses yang bermula dengan gambaran peringkat tinggi dan mewujudkan pandangan peringkat bawah yang dapat menentukan keselarasan antara modul-modul tersebut.

4.2 Asas unit kawalan

Pada asasnya, unit pemprosesan pusat(CPU) mengandungi beberapa elemen yang berfungsi. Elemen yang berfungsi bagi sesebuah CPU ialah

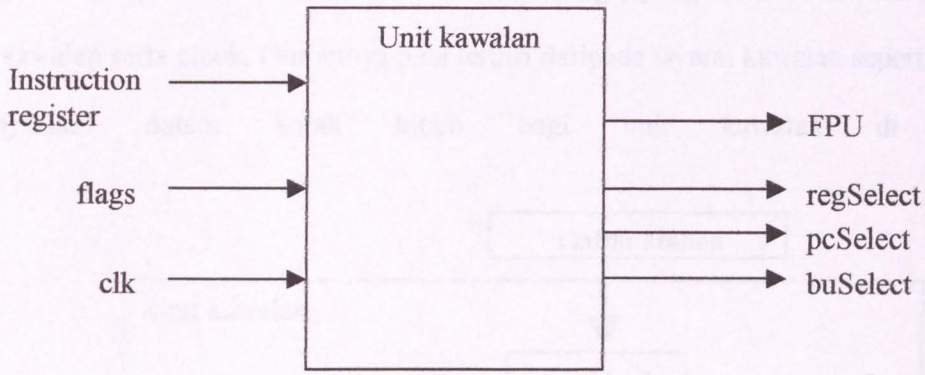
- Unit pelaksana(FPU)
- Perwakilan daftar (register)
- Set arahan (instruction set)
- Program counter
- Unit kawalan(CU)
- Unit cabang (branch unit)

Fungsi unit kawalan adalah untuk menyahkod arahan masukan dan menghasilkan isyarat output yang akan menentukan sesebuah operasi yang akan berlaku. Bagi unit kawalan, untuk menentukan fungsinya, mesti terdapat input yang membolehkannya untuk menentukan status bagi sistem dan output untuk membolehkannya mengawal segala tindakan dalam sistem. Kesemua ini merupakan spesifikasi luaran bagi unit kawalan. Input dan output bagi satu unit kawalan akan ditunjukkan di bawah:

- **Input unit kawalan**
  - **Clock-** berfungsi untuk menentukan masa bagi unit kawalan. Unit kawalan menyebabkan satu mikro-operasi berlaku dalam pada setiap denyutan clock(clock pulse).
  - **Daftar arahan(instruction register)-** opkod bagi arahan semasa akan digunakan untuk menentukan mikro-operasi yang akan berfungsi semasa kitar pelaksanaan.
  - **Bendera(Flags)-** flags diperlukan oleh unit kawalan bagi menentukan status bagi CPU dan operasi bagi FPU yang akan datang.

- **Output unit kawalan**:
  - Isyarat kawalan di dalam pemproses: ianya boleh dibahagikan kepada dua pula iaitu:
    - Satu isyarat kawalan yang menyebabkan data dihantar daripada daftar(register) kepada daftar yang lain
    - Menentukan spesifikasi bagi unit pelaksana(FPU) untuk berfungsi
  - Isyarat kawalan kepada bus kawalan: ianya juga dibahagikan kepada dua iaitu:
    - Isyarat kawalan kepada ingatan

- Isyarat kawalan kepada modul-modul input dan output



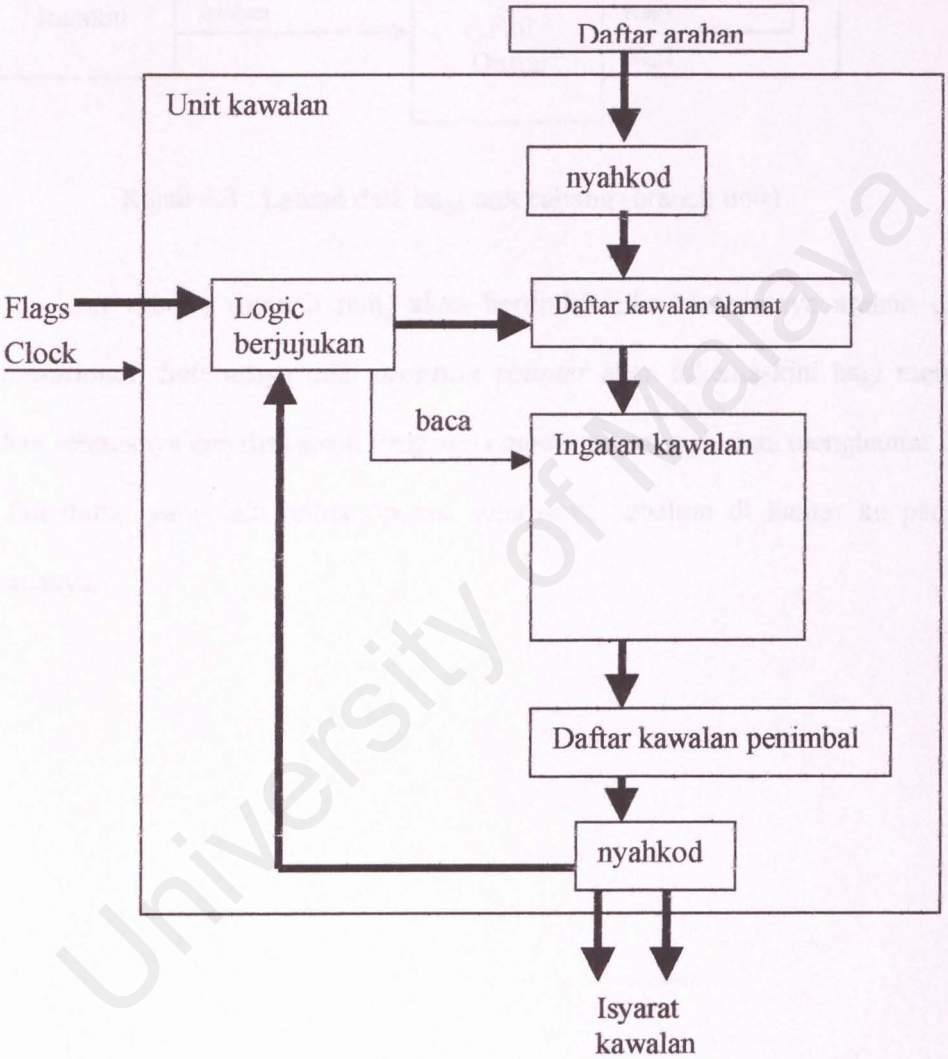
Rajah 4.1 : Kotak hitam (black box) unit kawalan

Terminal daftar(register) pada unit kawalan akan menentukan kemanakah isyarat kawalan akan dihantar. Output ini akan dihantar ke daftar-daftar yang tertentu iaitu:

- Daftar IF/ID
- Daftar ID/EX
- Daftar EX/MEM
- Daftar WB
- Program counter
- dan Unit cabang

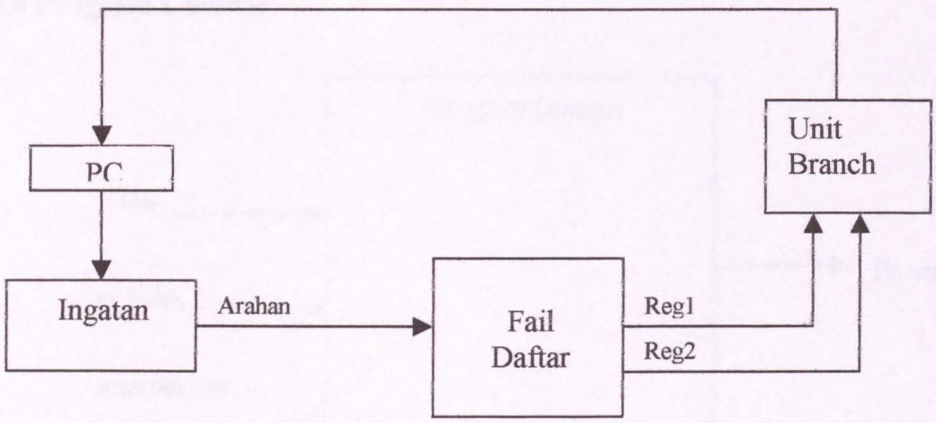
Isyarat kawalan FPU pula akan menentukan operasi yang akan dilakukan oleh unit pelaksana bagi titik apung. Daftar-daftar yang dinyatakan di atas dapat dilihat dalam hasil kerja rakan penulis. Terminal pcSelect akan menghantar data ke daftar *program counter* bagi pelaksaan arahan. Terminal buSelect pula akan menghantar data ke unit cabang bagi pelaksanaan arahan.

Gambarajah di bawah merupakan gambarajah blok bagi sebuah unit kawalan. Di mana input dan output masih sama iaitu terdiri daripada daftar arahan yang mengandungi opkod untuk ditafsirkan. Bendera (flags) yang menentukan status bagi unit kawalan serta clock. Outputnya pula terdiri daripada isyarat kawalan seperti yang ditunjukkan dalam kotak hitam bagi unit kawalan di atas



Rajah 4.2 : Gambarajah blok Unit kawalan

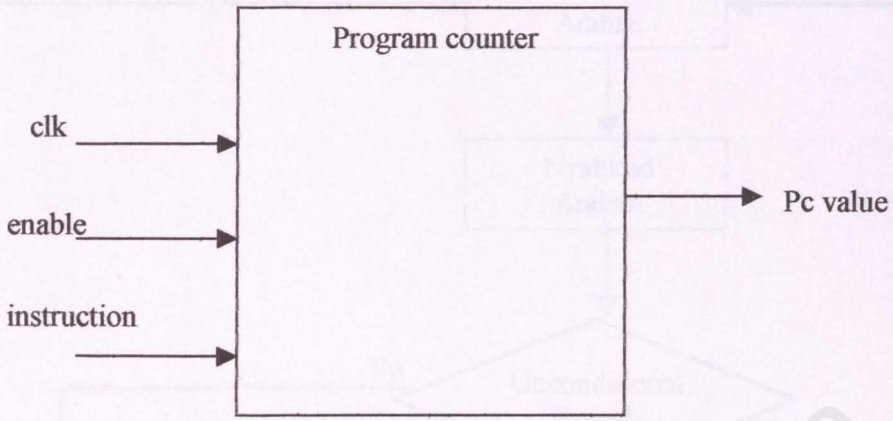
4.3 Unit cabang (Branch Unit)



Rajah 4.3 : Laluan data bagi unit cabang (branch unit)

Unit cabang (branch unit) akan bertindak jika terdapatnya arahan cabang *unconditional*. Seterusnya nilai *program counter* akan dikemaskini bagi menerima arahan seterusnya dan disimpan. *Program counter* seterusnya akan menghantar output ke fail daftar yang lain untuk operasi seterusnya sebelum di hantar ke peringkat seterusnya.

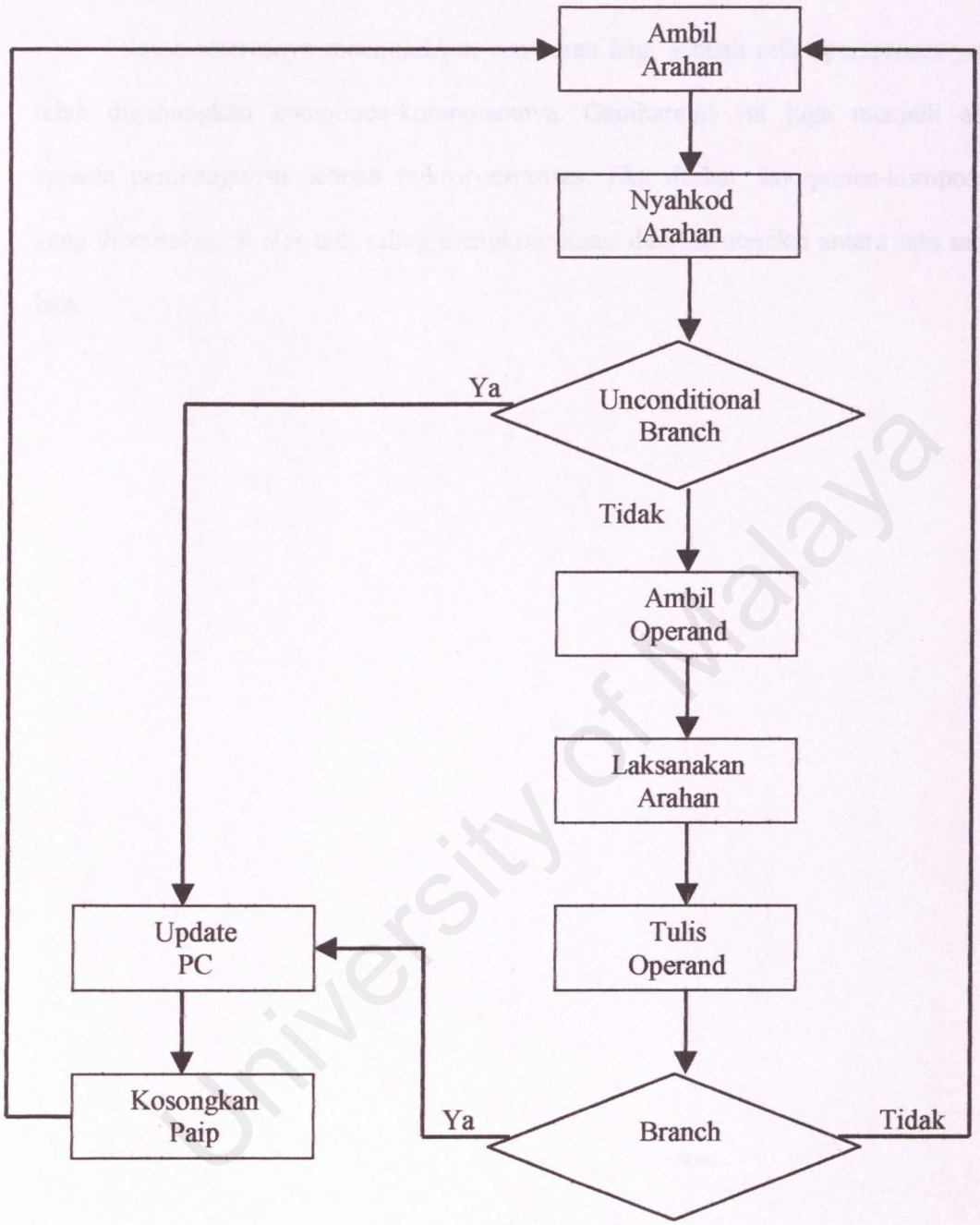
4.4 Program Counter



Rajah 4.4 : Kotak hitam bagi Program counter

Program counter akan menerima input daripada unit kawalan dan seterusnya akan menghantar nilai untuk disimpan dalam ingatan sebelum dihantar ke daftar-daftar khas untuk pelaksanaan seterusnya

4.5 Daftar Talian Paip

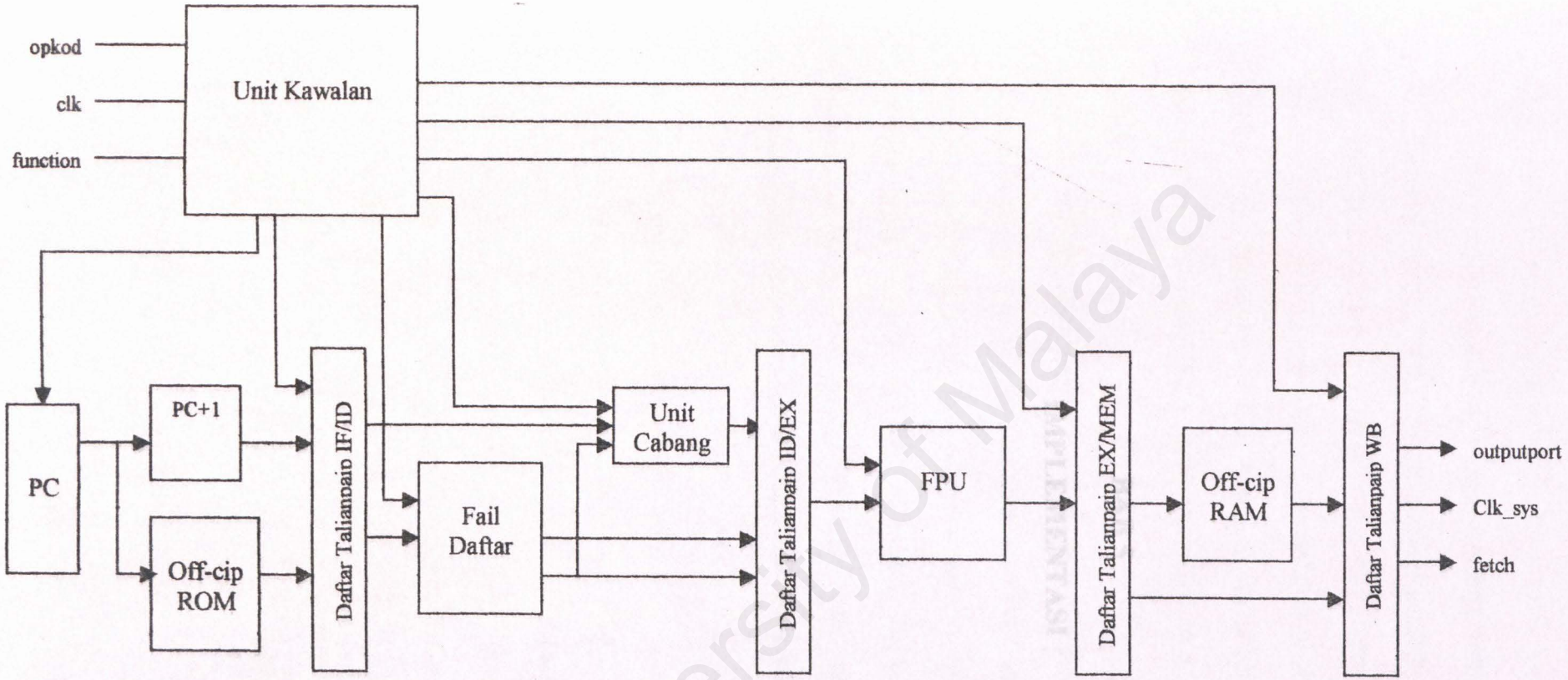


Rajah 4.5 : Aliran lima peringkat talian paip

Di dalam rajah aliran talian paip ini dapat dilihat dengan lebih jelas tentang fungsi unit cabang (*branch unit*). Dimana terdapat dua jenis cabang, iaitu cabang *conditional* dan *unconditional*.

4.6 Rekabentuk top level

Rajah seterusnya menunjukkan gambaran bagi sebuah mikropemproses yang telah digabungkan komponen-komponennya. Gambarajah ini juga menjadi asas kepada pembangunan sebuah mikropemproses. Jika dilihat, komponen-komponen yang diceritakan di atas tadi saling mengkoordinasi dan berinteraksi antara satu sama lain.



Rajah 4.6 : Gambarajah blok peringkat tinggi bagi mikropemproses

BAB 5

IMPLEMENTASI

5.1 Persekitaran Pembangunan

Fasa pengimplementasian sistem merupakan suatu proses di mana keperluan dan rekabentuk sistem ditukar ke dalam aturcara pengkodan. Daripada proses inilah, kita akan dapat melihat satu aliran bagi sesuatu input dan proses yang berlaku seterusnya kepada penghasilan output. Penghasilan ini adalah berdasarkan kepada kriteria yang dicadangkan pada peringkat rekabentuk dan pembangunan ini dilaksanakan dari semasa ke semasa sehinggalah menghasilkan satu sistem yang lengkap.

Persekitaran pembangunan kadang kala menghasilkan satu kesan semasa pembangunan sistem dijalankan. Keperluan perkakasan dan perisian perlu diikuti dan dipenuhi untuk meningkatkan kelajuan dan keupayaan semasa sistem ini dibangunkan di dalam proses pembangunan sistem. Perkakasan dan perisian memainkan peranan yang penting untuk menentukan keberkesanan sistem yang akan dibangunkan dan bersesuaian dengan persekitaran pengkomputeran pada masa kini.

5.2 Arahan yang diimplementasikan

Sebelum sesuatu pengkodan ataupun rekabentuk dilakukan seterusnya, penulis telah menetapkan beberapa arahan yang akan diimplemantasikan nanti. Arahan-arahan ini dipilih berdasarkan kepada beberapa nama yang terdapat pada pemproses yang terdapat pada masa kini. Walaubagaimanapun bukan semua arahan ini dapat disempurnakan oleh penulis dan ada diantaranya merupakan arahan yang boleh digunakan pada masa akan datang. Jadual di bawah menunjukkan beberapa arahan yang telah dipilih penulis dan disertakan dengan maksudnya sekali.

5.2.1 Arahan utama

Arahan	Maksud
NOP	Tiada operasi (No Operation)
ADD	operasi tambah
SUB	operasi tolak
MUL	operasi darab
NOT	Logik NOT
AND	Logik AND
OR	Logik OR
XOR	Logik XOR
NOR	Logik NOR

Jadual 5.1 : Arahan-arahan yang diimplementasikan

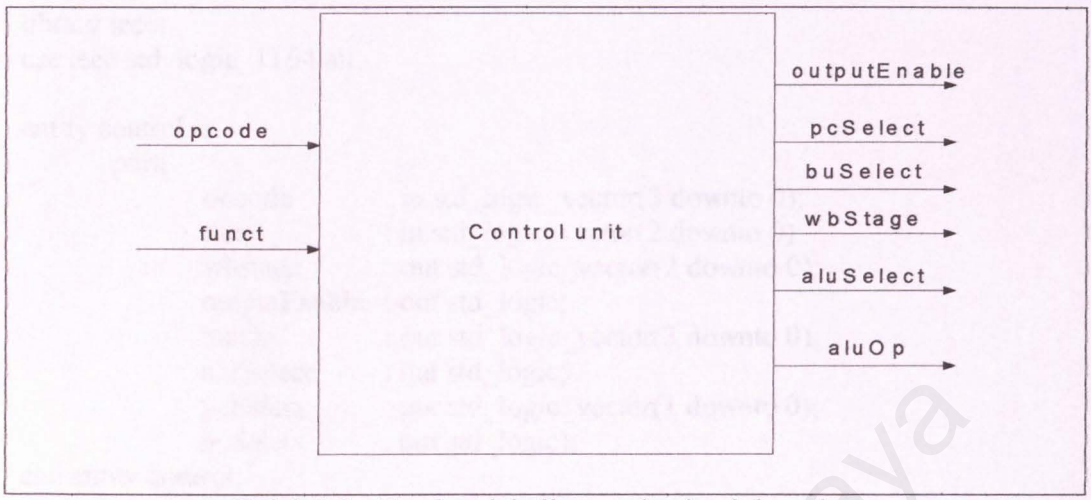
5.2.2 Arahan-arahan lain

BZ	Branch on Zero => cabang yang berfungsi apabila bernilai 0
BNZ	Branch Not Zero => cabang yang berfungsi bila bernilai selain 0
SLL	Shift Left Logic => logik anjakan ke kiri
SRL	Shift Right Logic => logik anjakan ke kanan
LW	Load Word
SW	Store Word

Jadual 5.2 : Arahan-arahan lain yang akan digunakan

5.3 Pengkodan

5.3.1 Unit Kawalan



Rajah 5.1 : Gambarajah diagram bagi unit kawalan

opcode	input 4 bit yang akan menentukan tindakan yang akan berlaku dalam unit kawalan
funct	Input 3 bit yang akan menentukan sebarang operasi yang berlaku
pcSelect	Output yang akan terus ke Progam counter
buSelect	Output bagi mengawal branch unit
wbStage	Output keluaran ke daftar writeback
aluSelect	Output bagi memilih ALU
aluOp	Output yang akan ke idex_c dimana akan dinyahkod untuk menentukan operasi yang berlaku

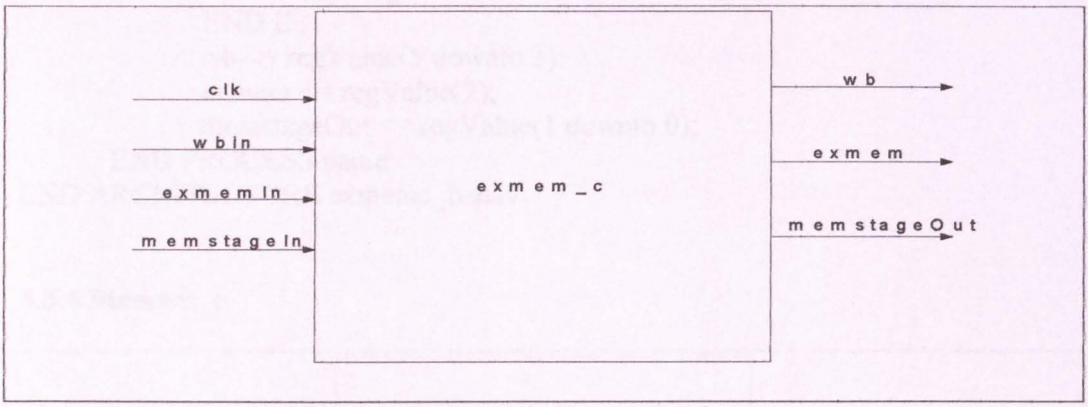
Jadual 5.3 : Penerangan bagi setiap port

Kod aturcara di bawah merupakan kod sumber bagi modul unit kawalan yang berfungsi seperti yang dinyatakan di atas.

```
library ieee;
use ieee.std_logic_1164.all;

entity control is
        port(
                opcode          : in std_logic_vector(3 downto 0);
                funct           : in std_logic_vector(2 downto 0);
                wbstage         : out std_logic_vector(2 downto 0);
                outputEnable    : out std_logic;
                aluOp           : out std_logic_vector(3 downto 0);
                aluSelect       : out std_logic;
                pcSelect        : out std_logic_vector(1 downto 0);
                buSelect        : out std_logic);
end entity control;

architecture control_behav of control is
begin
        name : process(opcode, funct) is
        begin
        case(opcode) is

                when "0000"=>
                        outputEnable <= '0';
                        aluSelect <= '0';
                        buSelect <= '0';
                        pcSelect <= "00";
                        wbstage <= "000";

                        case(funct) is
                                when "000"=>
                                        aluOp <= "1111";
                                when "001"=>
                                        aluOp<="0101";--ADD
                                when "011"=>
                                        aluOp<="0110";--SUB
                                when "101"=>
                                        aluOp<="1010";--MUL
                                when others =>
                                        aluOp<= "0100";--nop
                        end case;

                when "0001"=>
                        outputEnable <= '1';
                        aluSelect <= '0';
                        buSelect <= '0';
```

```
                pcSelect <= "00";
                wbstage <= "001";

                case(funct) is
                        when "000"=>
                                aluOp <= "0000";--AND
                        when "001"=>
                                aluOp<="0001";--OR
                        when "010"=>
                                aluOp<="0010";--XOR
                        when "011"=>
                                aluOp<="0011";--NOR
                        when "100"=>
                                aluOp<="0100";--NOT
                end case;

        when others =>
                outputEnable <= '0';
                buSelect <= '0';
                pcSelect <= "00";
                wbstage <= "100";
        end case;
    end process name;
end architecture control_behav;
```

5.3.2 Exmem_c



Rajah 5.2 : Gambarajah diagram bagi exmem_c

Clk	Kawal pemasaan
WbIn	input yang diperoleh daripada unit kawalan iaitu wbstage
exmemIn	input yang diperoleh daripada unit kawalan
memstageIn	input yang diperoleh daripada unit kawalan
Wb	Output yang akan merujuk kepada daftar memory write back
exmem	Output yang akan terus daftar bagi execution dan memory
memstageOut	Output yang akan masuk ke komponen daftar memory stage

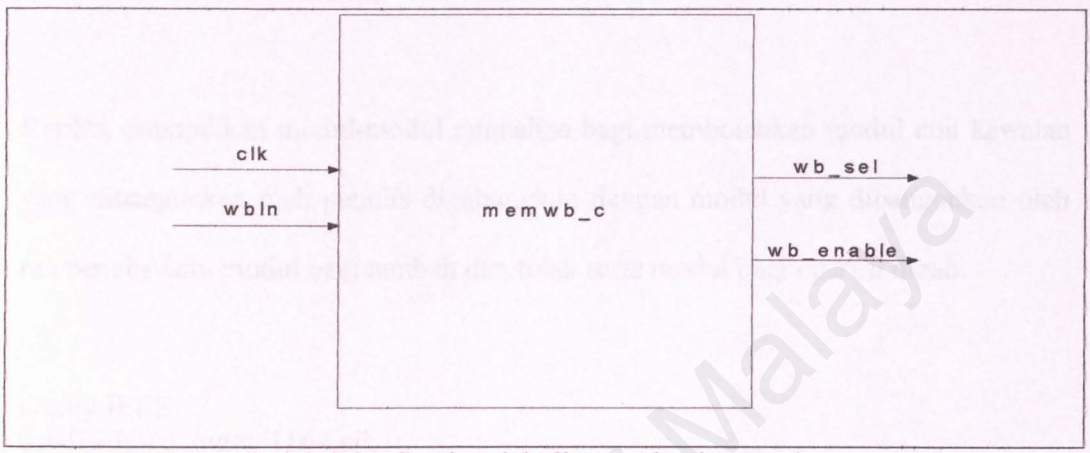
Jadual 5.4 : Penerangan bagi setiap port input dan output

Kod aturcara di bawah merupakan kod sumber bagi modul exmem_c.

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY exmem_c IS
PORT(
        clk             : IN STD_LOGIC;
        wbstage         : IN STD_LOGIC_VECTOR(2 downto 0);
        exmemIn         : IN STD_LOGIC;
        memstageIn      : IN STD_LOGIC_VECTOR(1 downto 0);
        wb              : OUT STD_LOGIC_VECTOR(2 downto 0);
        exmem           : OUT STD_LOGIC;
        memstageOut     : OUT STD_LOGIC_VECTOR(1 downto 0));
END ENTITY exmem_c;

ARCHITECTURE exmemc_behav OF exmem_c IS
BEGIN
        name : PROCESS(clk) IS
                VARIABLE regValue : STD_LOGIC_VECTOR(5 downto 0);
        BEGIN
                IF(clk='1') THEN
```

```
            regValue(5 downto 0) := wbstage(2 downto 0) & exmemIn &
            memstageIn(1 downto 0);
        END IF;
        wb <= regValue(5 downto 3);
        exmem <= regValue(2);
        memstageOut <= regValue(1 downto 0);
    END PROCESS name;
END ARCHITECTURE exmemc_behav;
```

5.3.4 Memwb_c



Rajah 5.3 : Gambarajah diagram bagi memwb_c

Clk	Kawal pemasaan
WbIn	merupakan input bagi memwb_c. Ianya diperolehi daripada unit kawalan iaitu wbstage
wb_sel	Output yang diperolehi adalah 2 bit seterusnya akan menjadi input bagi ke daftar bagi memory write back
wb_enable	Output 1 bit ini juga akan mengawal masukan bagi daftar memory write back

Jadual 5.4 : Penerangan bagi setiap port input dan output modul memwb_c

Kod aturcara di bawah merupakan kod sumber bagi modul memwb_c.

```
library ieee;
use ieee.std_logic_1164.all

ENTITY memwb_c IS
  PORT(clk          : IN STD_LOGIC;
       wbstage      : IN STD_LOGIC_VECTOR(2 downto 0);
       wb_sel       : OUT STD_LOGIC_VECTOR(1 downto 0);
       wb_enable    : OUT STD_LOGIC);
END ENTITY memwb_c;
```

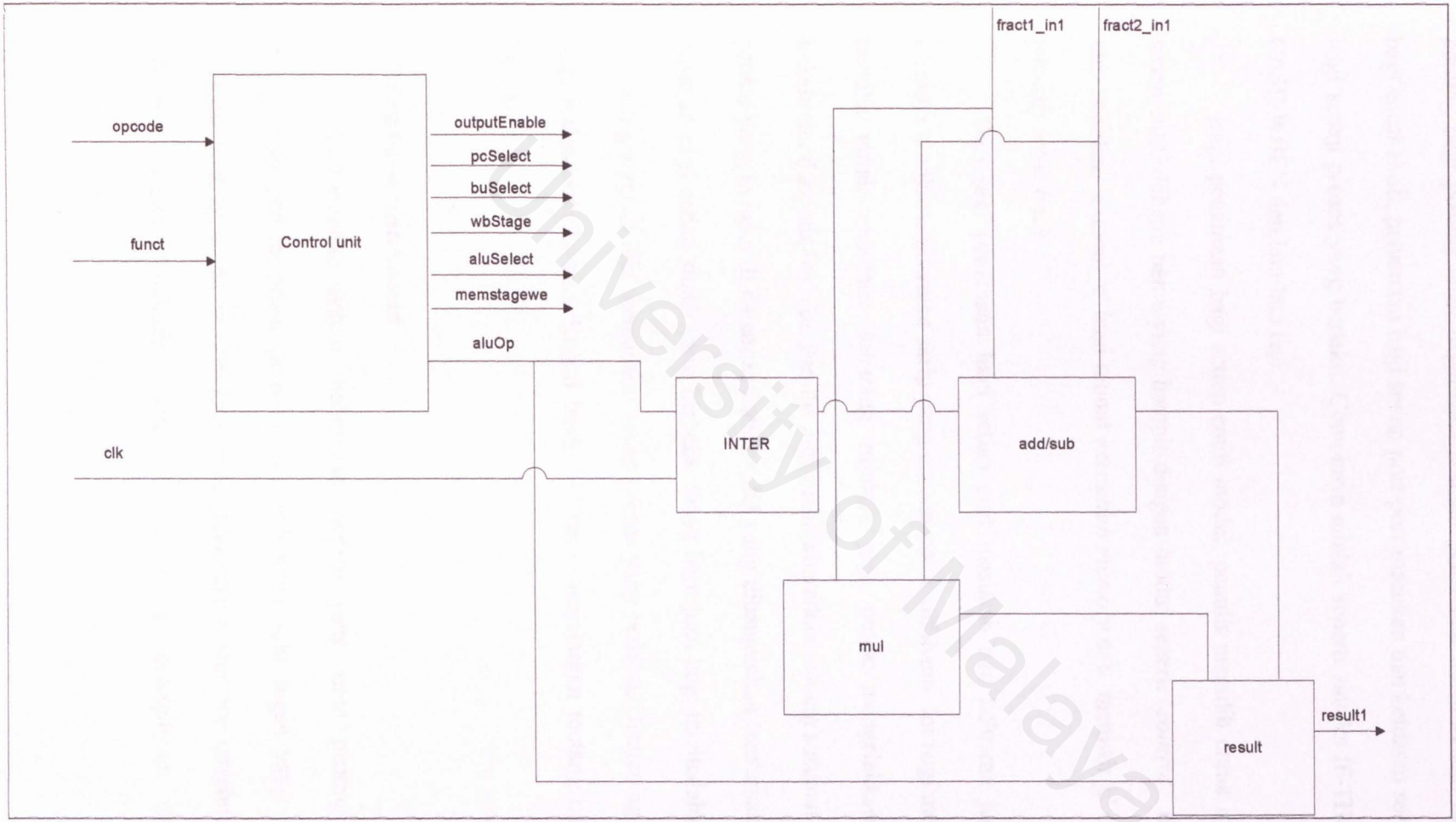
```
ARCHITECTURE memwbc_behav OF memwb_c IS
BEGIN
   name : PROCESS(clk,wbstage) IS
      VARIABLE regValue : STD_LOGIC_VECTOR(2 DOWNTO 0);
   BEGIN
      IF(clk='1') THEN
         regValue(2 DOWNTO 0) := wbstage;
      END IF;
      wb_sel <= regValue(2 DOWNTO 1);
      wb_enable <= regValue(0);
   END PROCESS name;
END ARCHITECTURE memwbc_behav;
```

Berikut ditunjukkan modul-modul tambahan bagi membolehkan modul unit kawalan yang dibangunkan oleh penulis digabungkan dengan modul yang dibangunkan oleh ran penulis iaitu modul bagi tambah dan tolak serta modul bagi operasi darab.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity toplevel is
   port (
         clk1           : in STD_LOGIC;
         opcode1        : in std_logic_vector(3 downto 0);
         funct1         : in std_logic_vector(2 downto 0);
         fract1_in1     : IN std_logic_vector (23 downto 0) ;
         fract2_in1     : IN std_logic_vector (23 downto 0) ;
         wbstage1       : out std_logic_vector(2 downto 0);
         outputEnable1 : out std_logic;
         aluSelect1     : out std_logic;
         pcSelect1      : out std_logic_vector(1 downto 0);
         memstagewe1 : out std_logic_vector(1 downto 0);
         buSelect1      : out std_logic;
         carry1         : out std_logic;
         result1        : out std_logic_vector (47 downto 0)
   );
end toplevel;

architecture toplevel_arch of toplevel is

signal s_mode          : STD_LOGIC_VECTOR(1 downto 0);
signal s_sum           : std_logic_vector(23 downto 0);
signal s_prod          : std_logic_vector(47 downto 0);
signal sin_aluOp       : std_logic_vector(3 downto 0);
```

```
component control
        port (
                opcode          : in std_logic_vector(3 downto 0);
                funct           : in std_logic_vector(2 downto 0);
                wbstage         : out std_logic_vector(2 downto 0);
                outputEnable    : out std_logic;
                aluOp           : out std_logic_vector(3 downto 0);
                aluSelect       : out std_logic;
                pcSelect        : out std_logic_vector(1 downto 0);
                memstagewe      : out std_logic_vector(1 downto 0);
                buSelect        : out std_logic
        );
end component;

component INTER
   port (
      clk       : in std_logic;
      in_aluOp  : in std_logic_vector(3 downto 0);
      mode      : out std_logic_vector(1 downto 0)

   );
end component;

component add_sub
   port (
     add        : IN    std_logic_vector(1 downto 0) ;
     fract1_in  : IN    std_logic_vector (23 downto 0) ;
     fract2_in  : IN    std_logic_vector (23 downto 0) ;
     carry      : OUT   std_logic ;
     sum        : OUT   std_logic_vector (23 downto 0)
   );
end component;

component mul
        port (
        clk     : IN    std_logic ;
        fract1  : IN    std_logic_vector (23 downto 0) ;
        fract2  : IN    std_logic_vector (23 downto 0) ;
        prod    : OUT   std_logic_vector (47 downto 0)
  );
end component;

component result
        port (
                clk             : in std_logic;
                addsub_res      : in std_logic_vector (23 downto 0);
                prod_res        : in std_logic_vector (47 downto 0);
                sel             : in std_logic_vector (3 downto 0);
                out_result      : out std_logic_vector (47 downto 0)
                        );
end component;
```

```
begin
U1: control port map (
                        opcode=>opcode1,
                        funct=>funct1,
                        wbstage=>wbstage1,
                        outputEnable=>outputEnable1,
                        aluOp=>sin_aluOp,
                        aluSelect=>aluSelect1,
                        pcSelect=>pcSelect1,
                        memstagewe=>memstagewe1,
                        buSelect=>buSelect1);

U2:  INTER port map (clk=>clk1,
                        in_aluOp=>sin_aluOp,
                        mode=>s_mode);

U3:  add_sub  port map (add=>s_mode,
                        fract1_in=>fract1_in1,
                        fract2_in=>fract2_in1,
                        carry=>carry1,
                        sum=>s_sum);

U4: mul port map (clk=>clk1,
                        fract1=>fract1_in1,
                        fract2=>fract2_in1,
                        prod=>s_prod);

U5: result port map (  clk=>clk1,
                        addsub_res=>s_sum,
                        prod_res=>s_prod,
                        sel=>sin_aluOp,
                        out_result=>result1);
end toplevel_arch;
```



Rajah 5.6 : Top level

5.4 Teknik Pengekodan

Di dalam membangunkan sebuah aturcara ataupun pengkodan yang baik, penulis telah merujuk kepada beberapa contoh kod sumber yang sedia ada. Apa yang penting bagi penulis ialah bagaimana sesuatu penamaan itu diberikan terutamanya bagi entiti-entiti, penamaan bagi setiap port-port masukan dan keluaran serta susunan bagi setiap proses yang berlaku. Contohnya adalah seperti sintaks IF-THEN-ELSE, CASE-WHEN dan lain-lain lagi.

Bagi penamaan bagi setiap entiti modul, penulis memilih untuk menamakan setiap entiti dengan nama yang hampir dengan modul seperti '*control*' bagi modul unit kawalan, 'exmem_c' bagi modul *execution memory* dan 'memwb_c' bagi modul *memory write back.*

Dari segi penamaan bagi setiap port masukan dan keluaran juga penulis memilih untuk memberikan skrip yang mudah untuk difahami. Ini bagi memudahkan penulis untuk membuat sebarang rujukan juga untuk mengelakkan sebarang kekeliruan daripada berlaku. Penulis juga menitikberatkan tentang susunan bagi setiap proses yang berlaku di dalam sesuatu modul yang dibangunkan terutamanya tentang susunan bagi setiap sitaks yang berbeza. Ianya bertujuan bagi memudahkan penulis ataupun pengguna lain memahami setiap proses yang berlaku. Disamping itu penulis juga meletakkan komen dengan tanda '--' bagi menjelaskan tentang operasi yang berlaku.

5.5 Pembangunan Sistem

Pembangunan sistem merupakan elemen yang amat penting di dalam persekitaran pembangunan perisian. Ia melibatkan satu tugas yang memainkan peranan penting di dalam menjana kebolehlaksanaan aturcara tersebut atau juga dikenali sebagai Pengkodan Sistem. Di dalam membangunkan aturcara ini,

pengkodan VHDL telah digunakan sepenuhnya sebagai asas bahasa yang digunakan di mana ia juga digunakan sebagai satu bahasa pengaturcaraan berstruktur. Dalam membangunkan satu aplikasi perisian, satu tugas dilaksanakan di dalam projek untuk mengurus semua fail-fail yang berbeza yang akan menghasilkan satu aplikasi apabila digabungkan. Kesemua modul-modul yang telah dibina ini ditersambungkan untuk menjadi satu aplikasi yang lengkap.

5.6 Pengujian Sistem

Proses pengujian mungkin merupakan satu bahagian yang kurang difahami di dalam projek pembangunan perisian dan kefungsiannya di dalam perisian. Selain daripada itu, pengujian juga merupakan suatu proses latihan penggunaan dan penilaian sistem dengan menggunakan manual yang telah disediakan oleh pembangun sistem tersebut. Proses ini adalah bertujuan untuk mengesahkan dan memenuhi keperluan pengguna serta untuk mengenalpasti perbezaannya di antara keputusan yang dijangka dengan keputusan yang sebenar. Beberapa strategi pengujian digunakan di dalam menguji aturcara mikropemproses(unit kawalan) ini iaitu pengujian modul, pengujian integrasi dan pengujian sistem.

5.6.1 Pengujian Modul

Proses pengujian modul yang dijalankan ini adalah untuk mengenalpasti dan mengesan adakah kesemua komponen dan fungsian di dalam sistem ini berfungsi dengan betul dan berkesan melalui jenis input yang dijangkakan daripada kajian rekabentuk komponen-komponen tersebut. Langkah pertama yang perlu dilaksanakan adalah mereka satu algoritma untuk fungsian tersebut dan menulis satu sumber pengkodan dengan menggunakan algoritma tersebut. Selepas itu, semak semula keseluruhan sumber pengkodan itu untuk memastikan algoritmanya berfungsi dengan

berkesan serta membandingkan kod-kod tersebut dengan gambarajah blok dan rekabentuknya supaya semua perkara-perkara yang berkaitan telah dilaksanakan. Langkah seterusnya adalah dengan melarikan dan mengkompil sumber pengkodan tersebut dengan menggunakan PeakFPGA untuk memaparkan keputusan dalam bentuk simulasi di mana akan dipaparkan dalam bentuk gelombang(*waveforms*) dan menghapuskan kesilapan sintaks yang berlaku sekiranya ada. Akhir sekali, kes-kes ujian dibangunkan untuk menunjukkan bahawa input telah ditukar dengan teliti kepada *output* yang benar-benar dikehendaki.

5.6.2 Pengujian Integrasi Modul

Setelah semua komponen telah diuji sepenuhnya di dalam proses pengujian unit dan berfungsi dengan betul memenuhi objektif yang dinyatakan, kesemua komponen ini digabungkan untuk menjadi satu sistem yang lengkap dan dapat dilaksanakan dengan berkesan. Proses pengujian integrasi ini sebenarnya boleh dikatakan sebagai satu proses untuk menguji dan menentukan samada semua modul komponen yang digabungkan mampu untuk berfungsi seperti yang dinyatakan di dalam fasa rekabentuk sistem. Pengujian adalah bertujuan untuk memastikan bahawa semua modul komponen ini adalah berfungsi dengan tepat. Sekiranya kesilapan ditemui, ia perlu diperbaiki dengan segera supaya objektif pembangunan sistem ini dapat dicapai dan dilaksanakan.

5.6.3 Pengujian Keseluruhan Sistem

Proses atau fasa pengujian sistem ini merupakan satu langkah pengujian yang terakhir dilaksanakan. Ia dilaksanakan bertujuan untuk memastikan aturcara yang dibangunkan ini dapat berjalan seperti yang dikehendaki bagi menghasilkan sebuah

mikropemproses. Aturcara ini diuji sepenuhnya samada memenuhi objektif kecekapan perlaksanaan yang spesifik di dalam pengujian perlaksanaan.

BAB 6

PENGUJIAN

Bab 06 : Pengujian

6.1 Pengenalan

Di dalam bab ini penulis akan menerangkan tentang bagaimana sesuatu modul-modul itu dibangunkan serta kaedah dan langkah-langkah yang diambil untuk menguji kebolehlaksanaan sesuatu modul seperti yang dikehendaki penulis

Pengujian merupakan langkah yang diambil bagi memastikan sesuatu modul itu dapat dilaksanakan mengikut arahan-arahan yang telah ditetapkan oleh penulis sendiri. Dengan melakukan pengujian ke atas modul yang telah dibangunkan, penulis dapat mengesan sebarang ralat yang berlaku pada modul iaitu ke atas kod-kod yang telah ditulis. Selain daripada mengesan ralat yang berlaku, teknik pengujian dapat memastikan tidak berlakunya sebarang kesalahan pada kod-kod VHDL yang telah ditulis dan memastikan ianya dapat beroperasi seperti yang dikehendaki oleh penulis.

6.2 Pembangunan Modul

Bagi memastikan sesuatu modul itu dapat berfungsi dengan lancar, penulis telah memilih untuk membahagikan suatu modul yang besar kepada modul-modul kecil bagi memudahkan pengesanan ka atas ralat yang berlaku. Langkah ini di diambil bertujuan untuk mempercepatkan pengesanan ke atas ralat yang berlaku ke atas kod-kod yang telah ditulis. Sebagai contoh penulis telah membahagikan satu modul unit kawalan (control unit) kepada beberapa blok mengikut input yang dimasukkan:

```
case(opcode) is
        when "0000"=>
                outputEnable <= '0';
                aluSelect <= '0';
                buSelect <= '0';
                pcSelect <= "00";

                case(funct) is
                        when "000"=>
                                aluOp <= "1111";--tiada operasi
                        when "001"=>
                                aluOp<="0101";--operasi tambah
                        when "011"=>
                                aluOp<="0110";--operasi tolak
                        when "101"=>
                                aluOp<="1010";--operasi darab
                        when others =>
                                aluOp<= "0100";--nop
                        end case;
```

Rajah 6.1 Contoh Aturcara Pengkodan bagi Unit Kawalan

Jika dilihat pada aturcara di atas, penulis telah memilih untuk mengesan ralat bagi input *opcode* bernilai "0000" terlebih dahulu sebelum kesemuanya digabungkan menjadi satu modul yang lengkap dan boleh beroperasi. Langkah yang diambil ini amat berkesan dalam memudahkan pengesanan ralat yang berlaku kerana dengan aturcara yang kecil maka sebarang ralat mudah dikesan dengan pantas dan pembetulan lebih efisien dapat dijalankan.

6.3 Pembangunan Test bench

Modul test bench dibangunkan secara berasingan dengan modul utama. Test bench bertujuan untuk menguji kebolehlaksanaan sesuatu modul yang telah dibangunkan. Dengan membangunkan test bench penulis dapat mengetahui sesuatu modul itu telah berfungsi seperti yang dikehendaki apabila ianya disimulasikan kemudiannya. Bagi aturcara seperti yang ditunjukkan di atas, contoh test bench yang di hasilkan adalah seperti berikut:

```
process
        begin
                opcode<= "0000";
                funct<= "000";
                wait for PERIOD;
                funct<="001";
                wait for PERIOD;
                funct<="011";
                wait for PERIOD;
                funct<="101";
                wait for PERIOD;
                funct<="110";
                wait for PERIOD;
```

Rajah 6.2 : Contoh Test Bench bagi Unit Kawalan dalam Rajah 6.1

di mana nilai input ditetapkan seterusnya kita akan dapat melihat keluaran output apabila telah disimulasikan nanti. Jika output yang dihasilkan nanti bertepatan dengan kehendak penulis, maka aturcara yang dihasilkan adalah tepat seperti yang dikehendaki.

6.4 Pengujian dengan PeakFPGA

Perisian PeakFPGA telah digunakan dalam membangunkan projek ini. Perisian ini menyediakan servis yang mudah bagi kegunaan pengguna. Penulis akan menerangkan tentang kemudahan ataupun ciri-cir yang disediakan dalam membangunkan modul-modul. Tiga menu utama yang penting dalam pengujian menggunakan perisian ini ialah pengkompil(compile), pautan(link) dan simulasi(simulate).

6.5 Pengkompil (compiler)

Perisian PeakFPGA menyediakan khidmat pengkompil untuk memudahkan penulis mengetahui jika terdapat ralat pada modul yang ditulis oleh penulis. Perisian ini akan mengeluarkan satu dialog yang memberitahu kepada pengguna tentang kesalahan yang terdapat pada aturcara yang telah ditulis. Antara contoh-contoh kesalahan yang sering berlaku ialah kesalahan dari segi sintaks, penamaan port yang tidak sama, panjang bit yang berbeza seperti yang telah diisytiharkan pada entiti (type mismatched) dan yang lain-lain lagi. Terdapat juga ralat yang kompleks yang mampu dikesan oleh perisian ini.

6.6 Link dan port map

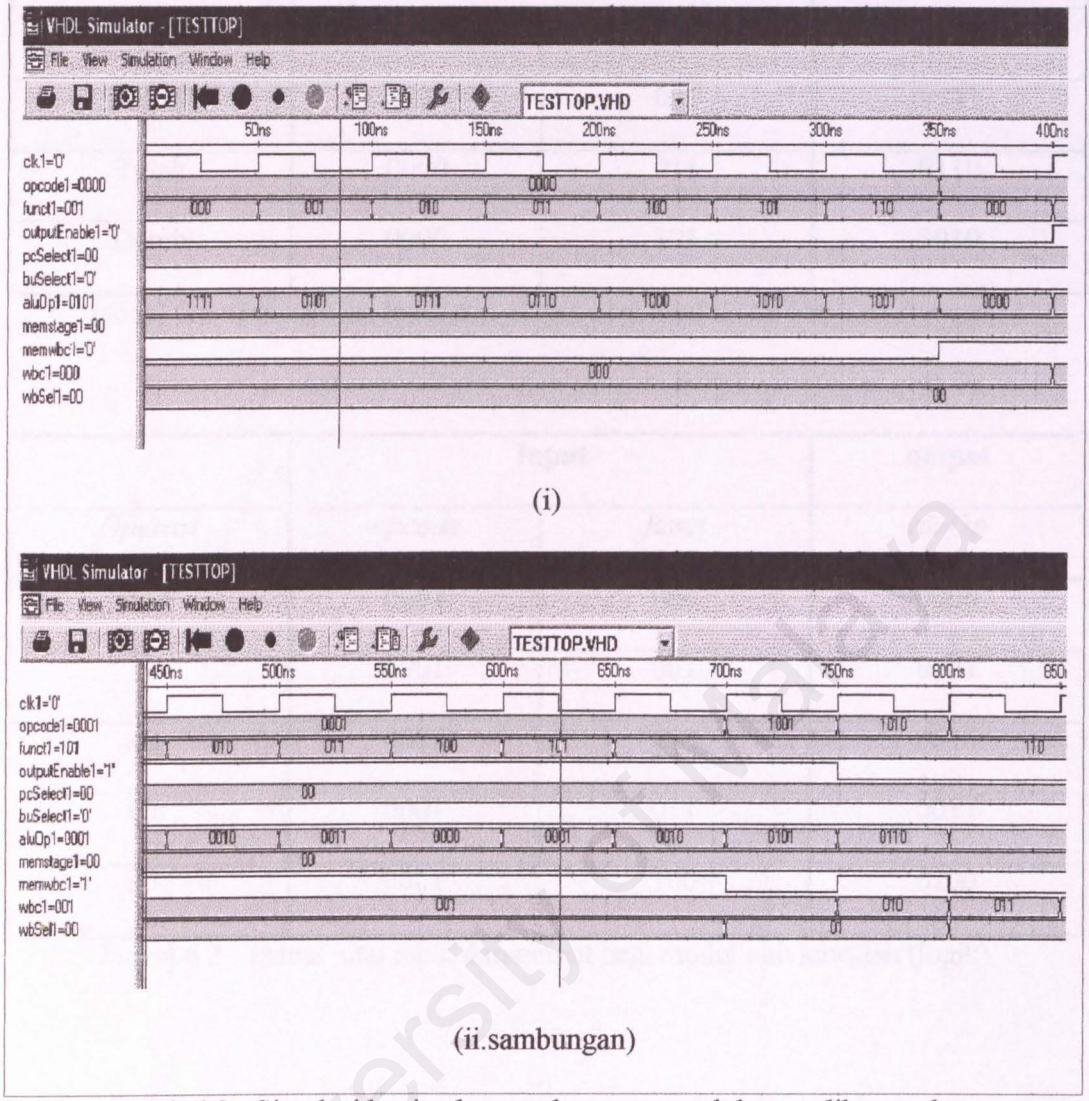
Link digunakan untuk membolehkan sesuatu modul itu diintegrasikan dengan modul yang lain. Sebelum itu, ia juga berfungsi untuk menghubungkan modul dengan test bench.

Port map merupakan satu kaedah di mana setiap modul yang dibangunkan tadi disatukan dengan menyediakan satu lagi modul port map yang tersendiri. Ianya melibatkan kesemua modul dan *top level*.

6.7 Simulasi

Merupakan antara langkah yang paling penting dalam memastikan sesuatu modul itu berfungsi seperti yang dikehendaki. Simulasi dapat menunjukkan aliran input, output ataupun signal yang diisytiharkan melalui bentuk gelombang atau 'waveforms'. Daripada simulasi ini, kita boleh memeriksa input yang dimasukkan dan juga output yang terhasil adalah seperti yang dijangkakan. Rajah-rajah di bawah menunjukkan gambarajah simulasi bagi modul-modul yang telah diuji.

6.7.1 Simulasi bagi gabungan kesemua modul yang dibangunkan



Rajah 6.3 : Simulasi bagi gabungan kesemua modul yang dibangunkan

Rajah simulasi di atas menunjukkan gambarajah simulasi utama di mana kesemua modul digabungkan dengan menggunakan teknik *'port map'* di mana kesemua modul disatukan dan menghasilkan rajah simulasi seperti di atas. Rajah simulasi yang dijanakan di atas merupakan gabungan modul unit kawalan, exmem_c, memwb_c dan idex_c.

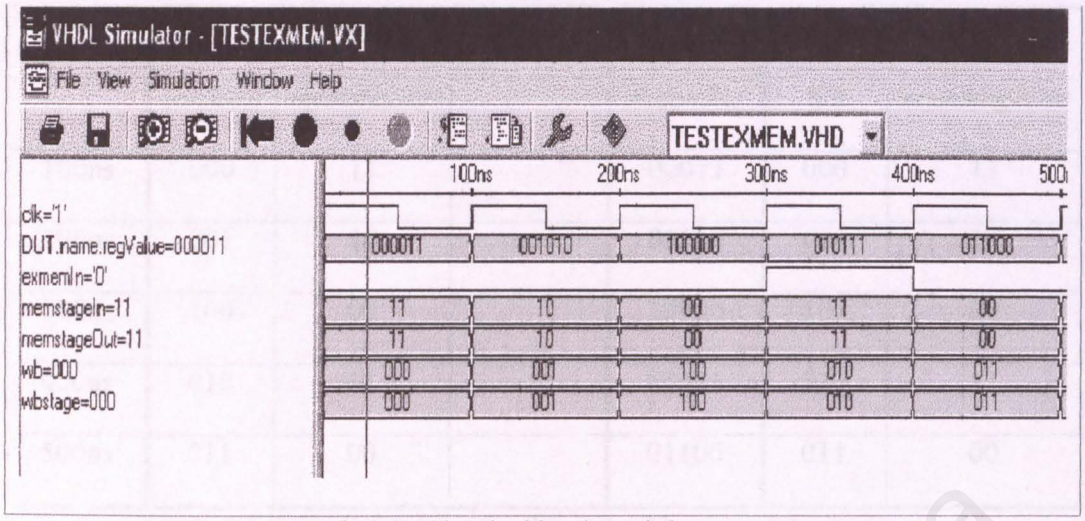
	Input		**output**
Operasi	*opcode*	*funct*	*aluOp*
tambah	0000	001	0101
Tolak	0000	011	0110
Darab	0000	101	1010

Jadual 6.1 : Jadual nilai input dan output bagi modul unit kawalan (operasi)

	Input		**output**
Operasi	*opcode*	*funct*	*aluOp*
and	0001	000	0000
or	0001	001	0001
xor	0001	010	0010
nor	0001	011	0011
not	0001	100	0100

Jadual 6.2 : Jadual nilai input dan output bagi modul unit kawalan (logik)

6.7.2 Simulasi bagi modul exmem_c



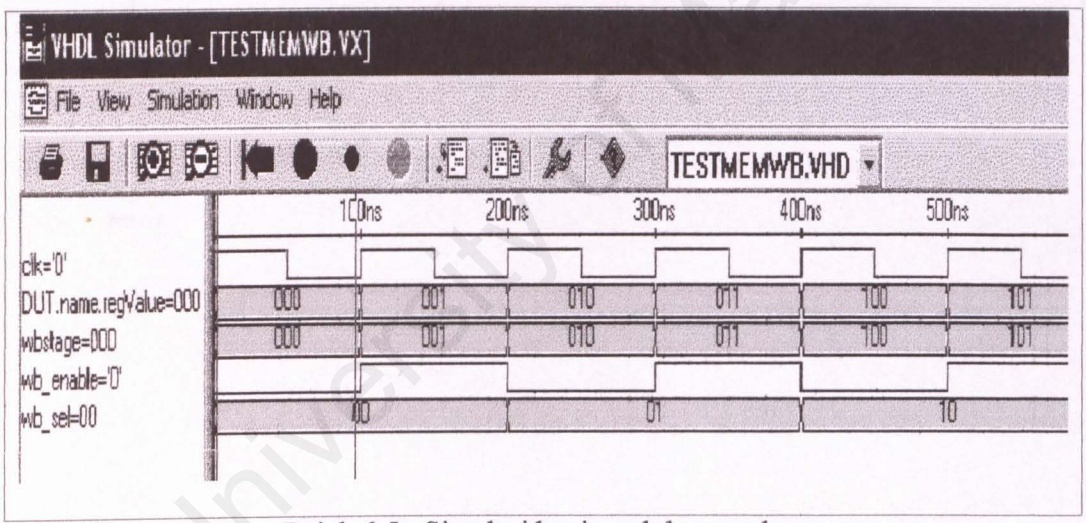
Rajah 6.4 : Simulasi bagi modul exmem_c

Rajah simulasi di atas merujuk kepada modul exmem_c. Rajah simulasi di atas menunjukkan input dan output yang dijanakan. Nilai input adalah merujuk kepada 'test bench' yang dijangka oleh penulis. Ini adalah kerana kesemua nilai input adalah daripada modul unit kawalan. Penulis menunjukkan bahawa modul ini dapat dijanakan dengan betul. Jika dilihat pada rajah simulasi di atas dapat dilihat nilai input iaitu wb, exmemIn dan memstageIn akan diumpukkan kepada satu nilai 'variable' iaitu regValue sebelum menjadi output. Sila rujuk jadual di bawah bagi membandingkan input dan output yang terdapat pada simulasi dan rujuk kepada modul aturcara yang terdapat pada apendiks.

	Input			Variable	Output	
	wbstage	*memstageIn*		*Regvalue*	*Wb*	*memstageOut*
100ns	000	11		00011	000	11
200ns	001	10		00110	001	10
300ns	100	00		10000	100	00
400ns	010	11		01011	010	11
500ns	011	00		01100	011	00

Jadual 6.3 : Jadual nilai input dan output bagi modul exmem_c

6.7.3 Simulasi bagi Modul memwb_c



Rajah 6.5 : Simulasi bagi modul memwb_c

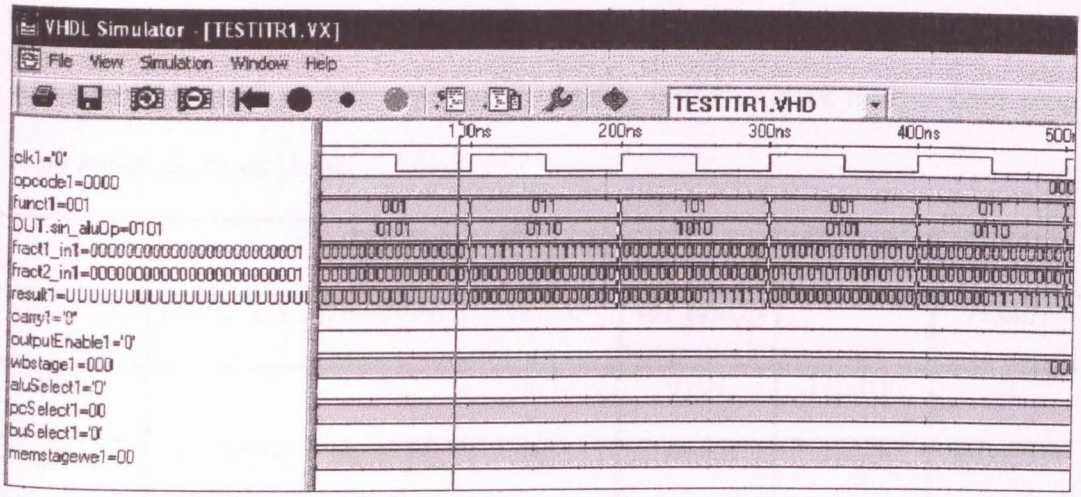
Rajah simulasi di atas menunjukkan pula simulasi bagi modul memwb_c. Seperti juga modul exmem_c, nilai yang diuji adalah berdasarkan test bench di mana inputnya adalah melalui jangkaan penulis berdasarkan modul unit kawalan. Apa yang ingin ditunjukkan di sini ialah bagaimana nilai input iaitu wbstage bernilai 3 bit diumpukkan dan menghasilkan 2 port keluaran. Jika dilihat pada jangkamasa 100ns

hingga 200 ns, nilai input ialah "001" dan seterusnya wb_sel akan mengambil nilai dari bit 2 hingga 1 iaitu "00" dan wb_enable pula mengambil nilai bit 0 iaitu "1" seperti yang dapat dilihat dalam jadual di bawah:

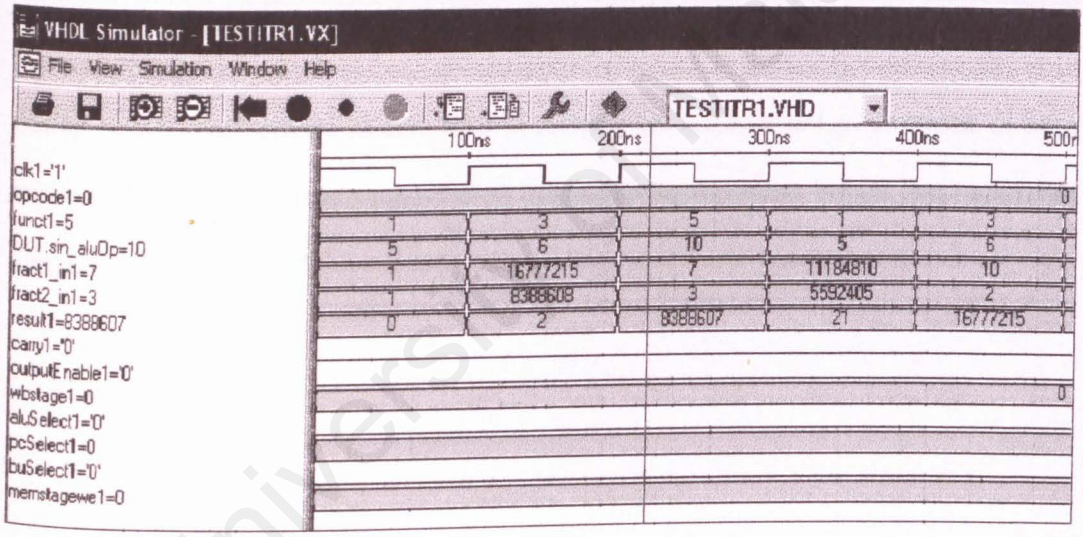
	Input	**variable**	**Output**	
	wbstage	*regValue*	*wb_sel*	*wb_enable*
100ns	000	000	00	0
200ns	001	001	00	1
300ns	010	010	01	0
400ns	011	011	01	1
500ns	100	100	10	0

Jadual 6.4 : Jadual nilai input dan output bagi modul memwb_c

6.7.4 Simulasi bagi setiap modul yang telah di *port map*



Rajah 6.6 : Simulasi bagi modul-modul yang telah di port map dalam bentuk binari



Rajah 6.7 : Simulasi bagi modul-modul yang telah di port map dalam bentuk desimal

Rajah 6.6 dan 6.7 menunjukkan rajah simulasi bagi penggabungan modul unit kawalan dengan modul tambah, tolak dan darab yang disediakan oleh rakan penulis. Penulis membahagikan rajah simulasi kepada dua iaitu dalam bentuk binari dan juga dalam bentuk desimal. Jika dilihat dalam rajah 6.6, didapati bahawa pada output result1 terdapatnya *delay* yang disebabkan oleh penggunaan *multiplexer* yang akan menyimpan jawapan terlebih dahulu dalam komponen sebelum output dipaparkan.

Jika dilihat pada jadual 6.1 dan rajah simulasi 6.6 ataupun 6.7, didapati bahawa apabila input funct bernilai '1' ataupun "001", maka operasi tambah akan berlaku. Keadaan ini berlaku apabila nilai opcode adalah "0000". Untuk melihat hasil yang lebih lanjut sila rujuk jadual di bawah:

	Input				**Operasi**	**Output**
	fract1_in1	*fract2_in1*	*funct*	*sin_aluOp*		*result1*
100ns	1	1	001	0101	Tambah	2
200ns	16777215	8388608	011	0110	Tolak	8388607
300ns	7	3	101	1010	Darab	21
400ns	11184810	5592405	001	0101	tambah	1677215

Jadual 6.5 : Input dan output bagi setiap operasi yang berlaku

Setiap result1 bagi jadual di atas adalah selepas berlakunya *delay*. Jawapan di baca anjak setiap 100ns. Perbandingan boleh dilakukan berdasarkan jadual dan juga rajah simulasi 6.6 ataupun 6.7.

BAB 7

PERBINCANGAN

Bab 07 : Perbincangan

Secara keseluruhannya, penulis telah berjaya mencapai tujuan utama untuk kajian kali ini iaitu bagi menghasilkan sebuah unit kawalan(control unit) yang dapat berfungsi bagi sesebuah mikropemproses. Walaubagaimanapun, disebabkan kekangan yang dihadapi semasa membangunkan projek ini, terdapat beberapa komponen yang tidak dapat dihasilkan semasa membangunkan mikropemproses kali ini. Dalam bab ini, penulis akan membincangkan tentang beberapa perkara yang dihadapi penulis semasa membangunkan projek ini antaranya termasuklah kebolehupayaan unit kawalan, masalah-masalah yang dihadapi berserta dengan penyelesaiannya, cadangan dimasa hadapan dan juga rumusan terhadap projek yang dijalankan sepanjang semester dua ini.

7.1 Kebolehupayaan

7.1.1 Kelebihan

Unit kawalan(control unit) merupakan komponen yang terpenting dalam sesebuah mikropemproses. Unit kawalan(control unit) juga merupakan komponen di mana data dan arahan keluar dan masuk ke dalam unit pemprosesan pusat. Walaubagaimanapun, hanya beberapa fungsi asas sahaja yang ditunjukkan dalam pembangunan unit kawalan bagi mikropemproses kali ini. Asas yang penting dalam pembangunan kali ini ialah untuk menunjukkan bagaimana sebuah unit kawalan berfungsi untuk mengawal operasi-operasi yang berlaku di dalam ALU(Arithmetic Logic Unit). Antara operasi-operasi yang terlibat adalah seperti operasi tambah(+), tolak(-) dan darab(x). Selain turut berfungsi untuk mengawal logik-logik seperti AND, OR, NOT, XOR dan lain-

lain lagi. Apa yang menarik lagi ialah unit kawalan berfungsi untuk mengawal operasi yang melibatkan nombor perpuluhan ataupun FPU(floating point unit) di mana modul bagi FPU ini dihasilkan oleh rakan penulis.

Jika dilihat di dalam bab pengujian, dapat dilihat di mana rajah-rajah yang menunjukkan rajah gelombang bagi setiap modul. Daripada rajah-rajah tersebut dapat diperhatikan bahawa unit kawalan yang dibangunkan ini berfungsi di mana setiap modul dapat diintegrasikan dengan sempurna.

7.1.2 Kekurangan

Unit kawalan yang dibangunkan penulis adalah terbatas dari segi fungsinya di mana ianya seperti yang dinyatakan dalam kelebihan di atas. Dengan fungsi yang terbatas ini ianya tentulah tidak dapat untuk menunjukkan sebuah miikropemproses yang lengkap. Banyak modul yang perlu dihasilkan lagi untuk sebuah mikropemproses yang lengkap. Antaranya ialah *'program counter'*, *'branch unit'* dan lain-lain lagi.

7.2 Masalah Pembangunan yang dihadapi

Di dalam membangunkan unit kawalan bagi sebuah mikropemproses ini, tentunya penulis tidak dapat lari daripada menghadapi sebarang masalah yang sukar untuk diatasi. Langkah penyelesaian perlu dirangka dengan segera supaya masalah-masalah ini boleh dipertimbangkan pada masa yang akan datang dalam membangunkan mikropemproses ini. Antara masalah yang dihadapi oleh penulis ialah:

7.2.1 Pengetahuan tentang VHDL

Pengetahuan penulis tentang VHDL adalah sangat terhad walaupun pernah mengikuti kursus VHDL sebelum ini. Jadi, penulis terpaksa merujuk kepada banyak sumber bagi memahami teknik dan cara untuk menghasilkan sebuah aturcara yang baik dalam VHDL. Penulis juga berhadapan dengan masalah untuk memahami sebarang kesilapan yang berlaku semasa pengujian dijalankan, terutamanya ketika proses mengkompil aturcara yang telah siap untuk diuji. Ini berlaku terutamanya untuk kesilapan yang tidak melibatkan sintaks dan jarang dijumpai oleh penulis seperti "aborting compile", amaran-amaran semasa mengkompil dan lain-lain lagi.

Penyelesaian

Penulis mengkaji bagi setiap kod-kod yang diperoleh daripada internet bagi memahirkan diri dengan VHDL. Penulis juga mempelajari bagaimana untuk menghasilkan sebuah teknik pengkodan yang baik melalui rujukan yang dibuat melalui internet dan juga buku. Hasilnya, walaupun masih kurang mahir tentang penggunaan VHDL, tetapi sekurang-kurangnya pengetahuan penulis terhadap VHDL adalah lebih baik sehingga dapat menghasilkan modul seperti unit kawalan yang telah dibangunkan oleh penulis. Penulis juga turut mendapatkan bantuan daripada penyelia projek dan juga rakan-rakan penulis yang terlibat dengan projek ini.

7.2.2 Perisian PeakFPGA

Perisian PeakFPGA merupakan sesuatu yang baru bagi penulis. Sebelum ini penulis menggunakan perisian Xillinx. Penulis terpaksa membiasakan diri

dengan perisian PeakFPGA terlebih dahulu sambil menghasilkan aturcara yang dikehendaki oleh penulis berkaitan dengan unit kawalan.

Penyelesaian

Penulis lebih kerap menggunakan perisian ini untuk memahirkan diri dengan fungsi-fungsi yang terdapat dalam perisian PeakFPGA ini. Penulis juga mengadakan perbincangan dengan rakan penulis untuk mengatasi masalah ini.

7.2.3 Pengintegrasian modul

Ini merupakan masalah utama yang dihadapi oleh penulis. Ini adalah kerana ianya melibatkan antara penulis dengan rakan penulis untuk menggabungkan modul-modul yang telah dihasilkan. Modul-modul yang telah dihasilkan harus digabungkan untuk membuktikan bahawa unit kawalan yang dihasilkan oleh penulis dapat berfungsi semasa pengujian dijalankan. Disebabkan masing-masing menghadapi masalah yang tersendiri terhadap penggunaan VHDL, jadi ianya memakan masa untuk menggabungkan modul-modul tersebut disamping kurangnya pengetahuan tentang teknik penggabungan modul.

Penyelesaian

Penulis telah mengadakan perbincangan dengan rakan penulis dan juga penyelia projek untuk menggabungkan modul-modul yang telah dibangunkan. Ini merupakan peringkat terakhir dalam projek penulis. Penulis juga turut menggunakan kaedah 'cuba jaya' untuk menggabungkan modul-modul ini.

7.2.4 Had masa

Untuk membangunkan sebuah unit kawalan yang berfungsi, pastinya memakan masa yang panjang. Disebabkan itu penulis hanya dapat menggabungkan modul penulis dengan ALU(Arithmetic Logic Unit) dan juga FPU(Floating Point Unit). Dalam masa lebih kurang tiga bulan untuk semester dua ini juga penulis harus mengkaji tentang penggunaan VHDL, memahirkan diri dengan perisian PeakFPGA disamping terpaksa juga menyiapkan tugasan-tugasan bagi kursus-kursus yang diambil sepanjang semester dua ini.

Penyelesaian

Penulis dan rakan penulis harus lebih peka terhadap pembahagian masa bagi membangunkan projek ini. Pembahagian masa yang lebih efisien harus dibentuk untuk membolehkan penulis memberi sepenuh tumpuan terhadap pembangunan projek. Pembahagian masa adalah penting untuk membolehkan penulis menyiapkan projek pada masa yang ditetapkan.

7.2.5 Sumber rujukan

Rujukan merupakan satu sumber penting di dalam membuat kajian dan membangunkan projek ini. Contohnya perpustakaan sendiri kurang menyediakan buku terhadap tajuk berkaitan VHDL dengan jumlah yang terlalu kecil ataupun sudah lama. Ini ditambah lagi dengan kerja-kerja pengubahsuaian yang dilakukan di perpustakaan utama sepanjang semester dua dan menyukarkan penulis untuk melakukan kajian terhadap tajuk berkaitan projek.

Penyelesaian

Memandangkan sumber rujukan dari buku yang berkurangan, penulis telah mengambil inisiatif untuk memperbanyakkan pencarian di internet sabagai salah satu sumber rujukan yang terpenting. Disamping itu pihak perpustakaan juga diminta untuk menyediakan lebih banyak rujukan berkaitan komputer terutamanya tentang tajuk berkaitan VHDL dan juga buku-buku edisi terkini juga amat diharap dapat disediakan oleh pihak perpustakaan.

7.2.6 Perkakasan dan Perisian

Masalah yang dihadapi penulis disini ialah berkaitan dengan perkakasan yang dimiliki oleh penulis tidak menepati spesifikasi yang diperlukan oleh perisian PeakFPGA yang memerlukan komputer berkuasa tinggi untuk membolehkan ianya dilarikan dengan lancar. Penggunaan komputer untuk penulis hanyalah terhad di makmal sahaja iaitu pada waktu pagi dan petang. Jadi sedikit sebanyak mengganggu usaha penulis dan rakan penulis dalam membangunkan projek mikropemproses ini.

Penyelesaian

Penulis membuat draf bagi setiap modul yang ingin dibangunkan. Ini bagi membolehkan penulis menggunakan setiap kemudahan di makmal dengan semaksima mungkin untuk menyiapkan modul-modul tersebut.

7.3 Cadangan masa hadapan

Sistem yang dibangunkan boleh dipertingkatkan dari semasa ke semasa mengikut kehendak dan perkembangan persekitaran sistem mikropemproses ini. Ini bagi memastikan bahawa projek ini tidak terhenti setakat ini sahaja dan sekurang-

kurangnya mikropemproses yang dibangunkan ini nanti dapat dilihat kefungsiannya dengan lebih baik daripada sekarang. Antara cadangan yang difikirkan boleh dipertingkatkan pada masa hadapan adalah:

1. Menambah modul 'program counter'

   Program counter berfungsi untuk mengekalkan trek bagi alamat memori dimana arahan seterusnya yang akan dilaksanakan disimpan. Alamat tersebut akan dihantar ke memori menerusi talian alamat dengan isyarat kawalan yang dihantar oleh talian kawalan. Isyarat ini akan mentafsirkan data yang dihantar daripada memori ke mikropemproses. Program counter akan menerima input daripada unit kawalan dan seterusnya akan menghantar nilai untuk disimpan dalam ingatan sebelum dihantar ke daftar-daftar khas untuk pelaksanaan seterusnya. Penambahan modul ini juga akan meningkatkan kefungsian bagi unit kawalan pada masa akan datang.

2. Menambah modul 'branch unit'

   Unit cabang akan melaksanakan arahan cabang di dalam kawalan program *(program control)*. Unit cabang biasanya merupakan satu alamat arahan. Apabila dilaksanakan arahan cabang akan menyebabkan perpindahan nilai alamat kepada *program counter*. Cabang terbahagi kepada dua iaitu *conditional* dan *unconditional*. Arahan cabang *unconditional* akan menyebabkan cabang akan di tetapkan alamatnya tanpa sebarang keadaan. Arahan cabang *conditional* pula akan menetapkan keadaan di mana cabang akan berfungsi sebagai contoh jika cabang tersebut bernilai positif atau cabang bernilai kosong (zero). Di sinilah unit kawalan akan menentukan samada unit cabang ini berfungsi pada nilai kosong (zero) ataupun sebaliknya.

3. Menambah modul daftar

Selain daripada dua modul di atas, penulis juga bercadang untuk menambahkan modul-modul daftar seperti modul *'instruction fetch'*, *'instruction decode'*, *'memory write back'*, dan juga *'execution'*. Jika kajian yang lebih mendalam dibuat, kesemua modul-modul ini akan saling berhubung kait antara satu sama lain dalam sesebuah mikropemproses yang lengkap.

4. Menambah modul ALU bagi logik-logik

Bagi penulis, modul bagi logik-logik seperti AND, OR, NOT, NOR dan XOR juga harus dibangunkan bagi menunjukkan kefungsian sesebuah unit kawalan selain daripada modul-modul yang telah disebutkan di atas. Logik-logik ini juga merupakan sesuatu yang asas dalam sesebuah mikropemproses. Selain daripada logik-logik di atas, penulis juga bercadang untuk menambahkan lagi keperluan bagi unit kawalan dengan penggunaan anjakan ke kiri dan juga ke kanan pada masa akan datang.

5. Talianpaip

Berfungsi untuk meningkatkan masukan arahan dengan memecahkan setiap arahan dengan kepada paras(stages) yang berlainan. Setiap paras ini diasingkan oleh daftar yang akan menyimpan keputusan yang sebelumnya dan ini membolehkan nilai yang betul digunakan pada paras yang berikutnya. Ini berlaku pada simulasi yang dilakukan pada rajah 6.7 dimana terdapatnya *delay*. Jadi dengan penggunaan talianpaip, masalah ini akan dapat diatasi.

Modul-modul yang dicadangkan di atas ini merupakan sebahagian daripada komponen yang berfungsi dengan kehadiran unit kawalan. Di mana unit kawalan akan mengawal sebarang operasi yang berlaku dalam modul-modul di atas. Modul seperti ALU, FPU, *program counter*, *branch unit* serta modul daftar di atas merupakan modul-modul yang saling berkaitan antara satu sama lain. Dengan penambahan modul-modul tersebut akan lebih menyerlahkan lagi tentang fungsi bagi unit kawalan serta membangunkan sebuah mikropemproses yang hampir sempurna.

7.4 Rumusan

Secara keseluruhannya, projek tahun akhir yang dibangunkan ini memenuhi objektif utama yang telah ditetapkan oleh penulis iaitu untuk menghasilkan sebuah unit kawalan yang dapat berfungsi untuk mengawal komponen-komponen lain bagi menjalankan sesuatu operasi. Apa yang penting bagi penulis di sini ialah pengalaman dan pengetahuan yang diperoleh sepanjang menjalankan projek ini. Penulis mendapati bahawa sekurang-kurangnya projek ini menjadikan penulis lebih mahir degan penggunaan VHDL terutamanya dari segi membangunkan perkakasan digital dengan menggunakan VHDL. Penulis juga dapat mempelajari teknik untuk menghasilkan kod-kod VHDL dengan lebih mendalam lagi. Penulis tidak dapat mengelakkan diri daripada sebarang kekangan yang berlaku. Kekangan-kekangan ini sedikit sebanyak merencatkan perjalanan projek ini. Tetapi dengan bantuan serta usaha daripada semua pihak akhirnya penulis dapat membangunkan sebuah unit kawalan yang mampu berfungsi.

Walaupun dalam projek ini penulis hanya dapat mengintegrasikan unit kawalan dengan Floating Point Unit, tetapi dengan cadangan yang dikemukakan tadi

diharap impian untuk menghasilkan sebuah mikropemproses yang lengkap akan tercapai dengan bantuan dan sokongan daripada pelbagai pihak.

APPENDIKS

Top level

```
library IEEE;
use IEEE.std_logic_1164.all;

entity toplevel is
   port (
        clk1            : in STD_LOGIC;
        opcode1         : in std_logic_vector(3 downto 0);
        funct1          : in std_logic_vector(2 downto 0);
        fract1_in1      : IN std_logic_vector (23 downto 0) ;
        fract2_in1      : IN std_logic_vector (23 downto 0) ;
        wbstage1        : out std_logic_vector(2 downto 0);
        outputEnable1 : out std_logic;
        aluSelect1      : out std_logic;
        pcSelect1       : out std_logic_vector(1 downto 0);
        memstagewe1 : out std_logic_vector(1 downto 0);
        buSelect1       : out std_logic;
        carry1          : out std_logic;
        result1         : out std_logic_vector (47 downto 0)
   );
end toplevel;

architecture toplevel_arch of toplevel is

signal s_mode : STD_LOGIC_VECTOR(1 downto 0);
signal s_sum   : std_logic_vector(23 downto 0);
signal s_prod  : std_logic_vector(47 downto 0);
signal sin_aluOp: std_logic_vector(3 downto 0);

component control
        port (
                opcode : in std_logic_vector(3 downto 0);
                funct : in std_logic_vector(2 downto 0);
                wbstage : out std_logic_vector(2 downto 0);
                outputEnable : out std_logic;
                aluOp : out std_logic_vector(3 downto 0);
                aluSelect : out std_logic;
                pcSelect : out std_logic_vector(1 downto 0);
                memstagewe : out std_logic_vector(1 downto 0);
                buSelect : out std_logic
        );
end component;

component INTER
        port (
                clk: in std_logic;
                in_aluOp: in std_logic_vector(3 downto 0);
                mode: out std_logic_vector(1 downto 0)

        );
```

```
 end component;

 component add_sub
        port (
                add : IN std_logic_vector(1 downto 0)  ;
                fract1_in : IN std_logic_vector (23 downto 0) ;
                fract2_in : IN std_logic_vector (23 downto 0) ;
                carry : OUT std_logic  ;
                sum : OUT std_logic_vector (23 downto 0)
        );
 end component;

 component mul
        port (
                clk  : IN    std_logic  ;
                fract1 : IN    std_logic_vector (23 downto 0) ;
                fract2 : IN    std_logic_vector (23 downto 0) ;
                prod : OUT    std_logic_vector (47 downto 0)
        );
 end component;

 component result
        port (
                clk : in std_logic;
                addsub_res : in std_logic_vector (23 downto 0);
                prod_res   : in std_logic_vector (47 downto 0);
                sel : in std_logic_vector (3 downto 0);
                out_result : out std_logic_vector (47 downto 0)
        );
 end component;

 begin
 U1: control port map (
                        opcode=>opcode1,
                        funct=>funct1,
                        wbstage=>wbstage1,
                        outputEnable=>outputEnable1,
                        aluOp=>sin_aluOp,
                        aluSelect=>aluSelect1,
                        pcSelect=>pcSelect1,
                        memstagewe=>memstagewe1,
                        buSelect=>buSelect1);

 U2:  INTER port map (
                        clk=>clk1,
                        in_aluOp=>sin_aluOp,
                        mode=>s_mode);

 U3:  add_sub  port map (
                        add=>s_mode,
                        fract1_in=>fract1_in1,
```

```
                    fract2_in=>fract2_in1,
                    carry=>carry1,
                    sum=>s_sum);

U4: mul port map (
                    clk=>clk1,
                    fract1=>fract1_in1,
                    fract2=>fract2_in1,
                    prod=>s_prod);

U5: result port map (
                    clk=>clk1,
                    addsub_res=>s_sum,
                    prod_res=>s_prod,
                    sel=>sin_aluOp,
                    out_result=>result1);
end toplevel_arch;
```

Modul bagi Unit kawalan

```
library ieee;
use ieee.std_logic_1164.all;

entity control is
        port(
                opcode : in std_logic_vector(3 downto 0);
                funct : in std_logic_vector(2 downto 0);
                wbstage : out std_logic_vector(2 downto 0);
                outputEnable : out std_logic;
                aluOp : out std_logic_vector(3 downto 0);
                aluSelect : out std_logic;
                pcSelect : out std_logic_vector(1 downto 0);
                memstagewe : out std_logic_vector(1 downto 0);
                buSelect : out std_logic);
end entity control;

architecture control_behav of control is
begin
        name : process(opcode, funct) is
                        begin
                        case(opcode) is

                                when "0000"=>
                                        outputEnable <= '0';
                                        aluSelect <= '0';
                                        buSelect <= '0';
                                        pcSelect <= "00";
                                        wbstage <= "000";
                                        memstagewe <= "00";

                                        case(funct) is
                                        when "000"=>
                                                aluOp <= "1111";
                                        when "001"=>
                                                aluOp<="0101";--signed addition
                                        when "011"=>
                                                aluOp<="0110";--signed subtraction
                                        when "101"=>
                                                aluOp<="1010";--signed multiply
                                        when others =>
                                                aluOp<= "0100";--nop
                                        end case;

                                when "0001"=>
                                        outputEnable <= '1';
                                        aluSelect <= '0';
                                        buSelect <= '0';
                                        pcSelect <= "00";
                                        wbstage <= "001";
```

```
                    memstagewe<= "00";
                    case(funct) is
--operasi untuk setiap logik berlaku seperti AND, OR, NOT, NOR, dan lain-lain

                        when "000"=>
                            aluOp <= "0000";
                        when "001"=>
                            aluOp<="0001";
                        when "010"=>
                            aluOp<="0010";
                        when "011"=>
                            aluOp<="0011";
                        when "100"=>
                            aluOp<="0000";
                        when "101"=>
                            aluOp<="0001";
                        when "110"=>
                            aluOp<="0010";
                        when others =>
                            aluOp<="0011";
                    end case;


                when others =>
                    outputEnable <= '0';
                    aluSelect<='0';
                    aluOp<="0000";
                    buSelect <= '0';
                    pcSelect <= "00";
                    wbstage <= "000";
                    memstagewe <= "00";
            end case;
            end process name;
end architecture control_behav;
```

Modul inter-perantara bagi unit kawalan dan modul add/sub

```
library ieee;
use ieee.std_logic_1164.all;

entity INTER is
   port (
        clk: in std_logic;
        in_aluOp: in std_logic_vector(3 downto 0);
        mode: out std_logic_vector(1 downto 0)

   );

end INTER;

architecture inter_arch of INTER is
 begin
  process(clk, in_aluOp)
   begin

        if clk='1' then
                if in_aluOp = "0101" then
                        mode<= "01"; --untuk tambah
                else
                        mode<= "10"; --untuk penolakan
     end if;
        end if;

        end process;
end inter_arch ;
```

Modul add/sub bagi operasi tambah dan tolak

```
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY add_sub IS
  PORT(
    add : IN    std_logic_vector(1 downto 0)  ;
    fract1_in : IN    std_logic_vector (23 downto 0) ;
    fract2_in : IN    std_logic_vector (23 downto 0) ;
    carry  : OUT   std_logic  ;
    sum : OUT   std_logic_vector (23 downto 0)
  );
END add_sub ;

ARCHITECTURE arch OF add_sub IS
   signal in1_int : std_logic_vector (24 downto 0) ;
   signal in2_int : std_logic_vector (24 downto 0) ;
   signal sum_int : std_logic_vector (24 downto 0) ;
```

```
BEGIN

    in1_int <= '0' & fract1_in;
    in2_int <= '0' & fract2_in;

    sum_int <= in1_int + in2_int WHEN (add = "01") else
            in1_int - in2_int;

    sum <= sum_int(23 downto 0);
    carry <= sum_int(24);

END arch;
```

Modul mul-bagi menjalankan operasi darab

```
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_signed.ALL;

ENTITY mul IS
  PORT(
    clk  : IN    std_logic  ;
    fract1  : IN    std_logic_vector (23 downto 0) ;
    fract2  : IN    std_logic_vector (23 downto 0) ;
    prod : OUT   std_logic_vector (47 downto 0)
  );
END mul ;

ARCHITECTURE arch OF mul IS
  SIGNAL prod1 : std_logic_vector(47 DOWNTO 0);
BEGIN
  PROCESS (clk)
  BEGIN
    IF clk'event AND clk = '1' THEN
      prod <= fract1 * fract2;
      --prod <= prod1;
    END IF;
  END PROCESS;
END arch;
```

Modul result bagi mengeluarkan satu ouput sahaja bagi setiap operasi yang berlaku

```
Library IEEE;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_misc.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

Entity result Is
        port (
```

```
                clk : in std_logic;
                addsub_res : in std_logic_vector (23 downto 0);
                prod_res   : in std_logic_vector (47 downto 0);
                sel : in std_logic_vector (3 downto 0);
                out_result : out std_logic_vector (47 downto 0)
                        );
end result;


Architecture  res_out of result Is

        signal result_addsub : std_logic_vector (47 downto 0);

begin

result_addsub <= "000000000000000000000000" & addsub_res;
PROCESS (clk)
        begin
        if clk'event AND clk = '1' then
                if (sel = "0101" or sel = "0110") then
                        out_result <= result_addsub;
                elsif (sel = "1010") then
                        out_result <= prod_res;
                end if;

        end if;

        end process;

end res_out;
```

Test bench bagi port map

```
library ieee;
use ieee.std_logic_1164.all;

entity testbnch is
end testbnch;

use work.all;

architecture stimulus of testbnch is

        component toplevel
                port (
                clk1            : in STD_LOGIC;
                opcode1         : in std_logic_vector(3 downto 0);
                funct1          : in std_logic_vector(2 downto 0);
                fract1_in1      : IN std_logic_vector (23 downto 0) ;
                fract2_in1      : IN std_logic_vector (23 downto 0) ;
                wbstage1        : out std_logic_vector(2 downto 0);
                outputEnable1 : out std_logic;
                aluSelect1 : out std_logic;
                pcSelect1 : out std_logic_vector(1 downto 0);
                memstagewe1 : out std_logic_vector(1 downto 0);
                buSelect1 : out std_logic;
                carry1          : out std_logic;
                result1         : out std_logic_vector (47 downto 0));
        end component;

constant PERIOD: time := 100 ns;

signal clk1             : STD_LOGIC;
signal opcode1          : std_logic_vector(3 downto 0);
signal funct1           : std_logic_vector(2 downto 0);
signal fract1_in1       : std_logic_vector (23 downto 0) ;
signal fract2_in1       : std_logic_vector (23 downto 0) ;
signal wbstage1         : std_logic_vector(2 downto 0);
signal outputEnable1 : std_logic;
signal aluSelect1       : std_logic;
signal pcSelect1        : std_logic_vector(1 downto 0);
signal memstagewe1      : std_logic_vector(1 downto 0);
signal buSelect1        : std_logic;
signal carry1           : std_logic;
signal result1          : std_logic_vector (47 downto 0);

        begin

        DUT : toplevel port map (clk1,opcode1,funct1,fract1_in1,
        fract2_in1,wbstage1,outputEnable1,aluSelect1,pcSelect1,memstagewe1,
        buSelect1,carry1,result1);
```

```
clock : process
begin
        clk1<='1';
        wait for PERIOD/2;
        clk1<='0';
        wait for PERIOD/2;
end process;

INPUT: process

begin
        opcode1<= "0000";
        funct1<= "001";
        fract1_in1 <= "00000000000000000000001";
        fract2_in1 <= "00000000000000000000001";
        wait for period;

        funct1<= "011";
        fract1_in1 <= "11111111111111111111111";
        fract2_in1 <= "10000000000000000000000";
        wait for period;

        funct1<= "101";
        fract1_in1 <= "00000000000000000000111";
        fract2_in1 <= "00000000000000000000011";
        wait for period;

        funct1<= "001";
        fract1_in1 <= "10101010101010101010101";
        fract2_in1 <= "01010101010101010101101";
        wait for period;

        funct1<= "011";
        fract1_in1 <= "00000000000000000001010";
        fract2_in1 <= "00000000000000000000010";
        wait for period;

        funct1<= "101";
        fract1_in1 <= "00000000000000000111010";
        fract2_in1 <= "00000000000000000001110";
        wait for period;

        funct1<= "001";
        fract1_in1 <= "00000000000000000001010";
        fract2_in1 <= "00000000000000000000010";
        wait for period;

        funct1<= "011";
        fract1_in1 <= "00000000000000100001010";
        fract2_in1 <= "00000000000000000000010";
        wait for period;
```

```
            funct1<= "101";
            fract1_in1 <= "00000000000000000011111010";
            fract2_in1 <= "00000000000000000000001010";
            wait for period;

            funct1<= "001";
            fract1_in1 <= "00000000000000000000001010";
            fract2_in1 <= "00000000000000000000001010";
            wait for period;

            opcode1<= "0001";
            funct1<= "000";
            wait for period;
            funct1<= "001";
            wait for period;
            funct1<= "010";
            wait for period;
            funct1<= "011";
            wait for period;
            funct1<= "100";
            wait for period;
            funct1<= "101";
            wait for period;
            funct1<= "110";
            wait for period;

            wait;
        end process;
end stimulus;
```

Modul exmem_c

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY exmem_c IS
PORT(
        clk : IN STD_LOGIC;
        wbstage : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
        exmemIn : IN STD_LOGIC;
        memstageIn : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        wb : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
        exmem : OUT STD_LOGIC;
        memstageOut : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END ENTITY exmem_c;

ARCHITECTURE exmemc_behav OF exmem_c IS
BEGIN
        name : PROCESS(clk) IS
                VARIABLE regValue : STD_LOGIC_VECTOR(5 DOWNTO 0);
        BEGIN
-- On the postive clock edge write the values to the register.
                IF(clk='1') THEN
                        regValue(5 DOWNTO 0) := wbstage & exmemIn &
memstageIn(1 DOWNTO 0);
                END IF;
                wb <= regValue(5 downto 3); -- Control lines for WB stage
                exmem <= regValue(2); -- Ouput enable
                memstageOut <= regValue(1 downto 0); -- Memory Stage Control
Lines
        END PROCESS name;
END ARCHITECTURE exmemc_behav;
```

Modul memwb_c

This entity describes the Memory / Write Back control register. Values are written to this register on the positive clock edge.

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY memwb_c IS
   PORT(clk : IN STD_LOGIC;
      wbstage : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
      wb_sel  : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
      wb_enable : OUT STD_LOGIC);
END ENTITY memwb_c;

ARCHITECTURE memwbc_behav OF memwb_c IS
BEGIN
   name : PROCESS(clk,wbstage) IS
```

```
    VARIABLE regValue : STD_LOGIC_VECTOR(2 DOWNTO 0);
  BEGIN

    IF(clk='1') THEN
      regValue(2 DOWNTO 0) := wbstage;
    END IF;
    wb_sel <= regValue(2 DOWNTO 1);
    wb_enable <= regValue(0);
  END PROCESS name;
END ARCHITECTURE memwbc_behav;
```

Rujukan

Buku.

Ajoy Kumar Ray, Kishor M Bhurchandi(2001), *INTEL MICROPROCESSOR Architecture, Programming and Interfacing*, McGraw Hill.

Bhasker J.(1999), *VHDL Primer*, 3rd ed, Prentice Hall, Inc

Morris Mano, M.(1982), *Computer System Architecture,* 2nd ed, Prentice Hall, Inc.

Morris Mano, M., Charles R. Kime (2000), Logic and Computer Design Fundamentals, 2nd edition, Prentice Hall, Inc.

William Stallings, *Computer Organization and Architecture: Designing for performance,* 5th ed, Prentice Hall, Inc

Internet

1. http://www.intel.com
2. http://www.opencores.org
3. http://www.ee.gatech.com
4. http://www.ece.rice.edu
5. http://www.gmvhdl.com
6. http://www.computer.org
7. http://peakFPGA.com