Lista 1

*Introdução a Arquitetura de Computadores*

1. **O QUE HÁ POR TRÁS DO SOFTWARE DE APLICAÇÃO?**

R: Para que um programa (sua aplicação) rode corretamente é preciso que ela se comunique de algum modo com o hardware. Existem nesse meio termo vários softwares (chamados softwares de sistema) que são organizados hierarquicamente. Dois exemplos importantes de software de sistema é o sistema operacional e o compilador. Dessa forma, embaixo de uma aplicação existem softwares que tratarão de converter as funções e procedimentos e dados da aplicação em números binários para que o hardware compreenda.

1. **QUE LINGUAGEM O COMPUTADOR ENTENDE?**

R: O computador trabalha apenas com sequências de números binários 0 e 1 que ativarão ou desligarão os transistores que o compõe. Programadores notaram que alguns comandos se repetiam várias vezes num código binário e resolveram simplificar as coisas utilizando palavras de fácil memorização que pudesse substituir os 0 e 1, mas que não exigissem muito processamento para que fossem convertidos para binário.

A linguagem assembly é esse código, ela utiliza mnemônicos que simbolizam certos códigos (instruções) em binário. Assim a escrita de um conjunto de instruções para a máquina realizar pode ser feito de forma mais simples e muitos erros podem ser evitados. Isso é devido ao fato de que ao invés de escrever vários 0 e 1s escreveremos letras e “palavras” que serão apenas substituídas por seus equivalentes em binários pelo programa Assembler, o interpretador dessa linguagem que não faz nada mais que olhar uma tabela com os códigos binários de cada mnemônico.

1. **QUAIS OS COMPONENTES BÁSICOS DO COMPUTADOR DE VON NEUMANN?** R: Basicamente o computador de Von Neumann inclui uma CPU (onde encontramos a unidade de controle e a unidade lógica-aritmética) uma memória principal, dispositivos de entrada e saída. Algumas concepções do modelo incluem registradores (incluindo registradores especiais como o PC que registra a próxima instrução a ser executada e outros registradores de estado) e o barramento (de dados e entrada e saída).
2. **DEFINA ARQUITETURA DO COMPUTADOR**

R: Também conhecido como Arquitetura do Conjunto de instruções, pode-se dizer que é uma interface abstrata entre o hardware e o software de nível mais baixo de uma máquina que incluem todas as informações necessárias para escrever um programa em linguagem de máquina e fazê-lo funcionar. Isso incluirá informações sobre os mais diversos módulos que compõe esse computador. De forma mais simplificada a Arquitetura é como se fosse um manual de instruções do hardware que informará ao programador as ferramentas que ele tem para realizar fazer seu software rodar. Por exemplo, a arquitetura contém as instruções que um determinado computador é capaz de realizar, desse jeito o programador adaptará seu software à utilizar aquela instrução.

1. **CONSIDERE UM MICROPROCESSADOR HIPOTÉTICO, CUJAS INSTRUÇÕES DE 32 BITS SÃO COMPOSTAS DE DOIS CAMPOS: O PRIMEIRO BYTE CONTÉM O CÓDIGO DE OPERAÇÃO E OS DEMAIS CONTÊM UM OPERANDO IMEDIATO OU UM ENDEREÇO DE OPERANDO. QUAL A CAPACIDADE MÁXIMA DE MEMÓRIA ENDEREÇÁVEL DIRETAMENTE (EM BYTES)?**

R: 24 bits para endereço. Ou seja,2^24 possíveis locais de endereço. A memória é de 32 bits, logo cada uma dos possíveis locais representa 4 bytes. Isso perfaz 2^26 bytes, 64MBytes.

*Conjunto de instruções Neander/Ramsés*

1. **FAÇA VÁRIOS PROGRAMAS DIFERENTES QUE ZEREM O ACUMULADOR.**

ORG 00H

LDA 80H

NOT

AND 80H

STA 80H

HLT

ou

LDA 80h

NOT

ADD 85h

STA 81h

LDA 82h

ADD 80h

HLT

1. **ESCREVA UM PROGRAMA QUE DETERMINE QUAL A MAIOR DE 3 VARIÁVEIS POSITIVAS DE 8 BITS ARMAZENADAS NAS POSIÇÕES DE MEMÓRIA 80H A 82H. O RESULTADO (A MAIOR VARIÁVEL) DEVE APARECER NA POSIÇÃO DE MEMÓRIA 83H.**

ORG 00h

LDA 80H

NOT

ADD 85H

ADD 81H

JN 20H

LDA 81h

NOT

ADD 85H

ADD 82H

JN 40H

LDA 82h

STA 86H

HLT

ORG 20h

LDA 80h

not

add 85h

ADD 82h

JN 30h

LDA 82H

STA 86H

HLT

ORG 30h

LDA 80H

STA 86h

HLT

ORG 40H

LDA 81H

STA 86H

HLT

1. **LIMPEZA DE UMA ÁREA DE MEMÓRIA: FAÇA UM PROGRAMA PARA ZERAR 32 POSIÇÕES DE MEMÓRIA CONSECUTIVAS NA MEMÓRIA. O ENDEREÇO INICIAL DESTA ÁREA É FORNECIDO NA POSIÇÃO 80H. É GARANTIDO QUE O ENDEREÇO INICIAL ESTÁ ENTRE 82H E DCH.**

020 81 LDA 81

2 A0 18 JZ 18

430 83 ADD 83

610 81 STA 81

820 80 LDA 80

A10 F STA F

C20 84 LDA 84

E10 89 STA 89

1020 80 LDA 80

1230 82 ADD 82

1410 80 STA 80

1680 0 JMP 0

18F0 HLT

1. **EXPLIQUE COMO A REPRESENTAÇÃO EM COMPLEMENTO DE 2 É USADA PELA ULA DO PROCESSADOR NEANDER.**

O complemento de 2 ajuda o neander a trabalhar com número negativos. Dessa forma com o uso de 8 bits, 7 deles representam o número e 1 o sinal. Assim o NEANDER poder somar/subtrair número no intervalo -128 até +127

1. **A PARTIR DO PROGRAMA DE MULTIPLICAÇÃO DE DOIS INTEIROS POSITIVOS, FEITO NA SALA DE AULA, PROPONHA UMA ESTRATÉGIA QUE PERMITA TAMBÉM MULTIPLICAR NÚMEROS NEGATIVOS.**

Antes de realizar a multiplicação, inicie uma posição de memória com o valor zero. Se o primeiro número for negativo, incremente a variável. Se o segundo número for negativo, decremente a variável. Se qualquer um dos dois for positivo, a variável não é atualizada. Se a posição de memória for 1, o resultado será negativo. Se for 0 será positivo. Calcula-se a multiplicação de forma normal, utilizando o código anterior e depois apenas troca-se o sinal do resultado caso necessário.

1. **FAÇA UM PROGRAMA PARA IDENTIFICAR SE UM NÚMERO É NEGATIVO OU POSITIVO.**

ORG 00H

LDA 80H

JN 10H

LDA 82H

STA 83H ;83H = 82H IF 80H>0

HLT

ORG 10H

LDA 81H

STA 83H ;83H = 81H IF 80H<0

HLT

1. **CONSIDERANDO O PROCESSADOR RAMSÉS, QUAL SERIA O CÓDIGO BINÁRIO PARA AS INSTRUÇÕES:**

ADD B #01

STR A 80,I

JMP 10

Considere os números em hexadecimal.

0011 0110 0000 0001

0001 0001 1000 0000

1000 0000 0001 0000

*Organização do processador Neander*

**1) DESCREVA O EFEITO DE UMA FALHA “STUCK-AT-0” (OU SEJA, INDEPENDENTE DO QUE DEVERIA SER, O SINAL É SEMPRE 0) PARA OS SINAIS MOSTRADOS A SEGUIR, NO CAMINHO DE DADOS DO PROCESSADOR NEANDER. QUE INSTRUÇÕES, SE HOUVER, NÃO FUNCIONARÃO CORRETAMENTE? EXPLIQUE POR QUÊ.**

**a) CargaAC**

Uma falha no “Carga AC” impediria que qualquer dado apresentado nas entrada do acumulador fosse passado para dentro dele. Isso impediria todas as instruções que envolvem salvar no acumulador de funcionar, LDA obviamente. ADD, OR, AND, NOT,

**b) CargaRDM**

Uma falha no “Carga RDM” impediria a leitura e escrita de dados na memória de funcionar. Isso porque aoescrever dados na memória colocamos esses dados na entrada de RDM e damos uma carga RDM. O mesmo aconteceria para uma leitura. Então as instruções que envolvem troca de dados com a memória seriam impedidas de funcionar. Isto é todas as instruções, pois é sempre necessário fazer a busca da próxima instrução a ser executada.

**c) SEL**

O multiplexador SEL ajuda a alternar a origem do endereço de uma instrução ou operando. Definindo se o endereço virá do PC ou do RDM. Uma falha nesse sinal impediria a atualização do endereço da instrução ou operando. Como quando select = 0 a instrução vem do PC, qualquer instrução com operando não poderia ser executada visto que o endereço do operando geralmente vem direto do RDM. As instruções que não funcionariam seriam: STA, LDA, ADD, OR,AND,

**d) AND**

O sinal AND em 0 simplesmente impediria essa operação de ser executada na ULA. Apenas a instrução AND seria afetada.

**e) Write**

O sinal Write permite que a memória entenda que o que está no RDM deve ser salvo na memória. Sem essa instrução não seria possível para a memória saber quando seria necessário fazer a escrita dos dados, logo a instrução que seria diretamente afetada seria a STA.

**2. CONSIDERE A FIGURA QUE MOSTRA A ORGANIZAÇÃO INTERNA DO PROCESSADOR NEANDER. EXPLIQUE AS ETAPAS DE EXECUÇÃO DA INSTRUÇÃO STA END INDICANDO OS BLOCOS E SINAIS ENVOLVIDOS EM CADA FASE.**

|  |  |
| --- | --- |
| SEL = 0  Carga REM | REM<-PC |
| Read  INC PC  Carga RDM | RDM<-MEM(PC) |
| Carga RI | RI<-RDM |
| Decodificação |  |
| SEL = 0  Carga REM | REM<-PC |
| Read  INC PC  Carga RDM | RDM<-MEM(PC) |
| Carga REM | REM<-RDM |
| Carga RDM | RDM<-AC |
| Write | MEM(end)<-RDM |

**3. DESEJAMOS ACRESCENTAR A INSTRUÇÃO SLA (SHIFT LEFT ACCUMULATOR – DESLOCANDO UM BIT À ESQUERDA NO ACUMULADOR) AO CAMINHO DE DADOS DO PROCESSADOR NEANDER. FAÇA AS INCLUSÕES E MODIFICAÇÕES NECESSÁRIAS NO CAMINHO DE DADOS PARA QUE ESTA INSTRUÇÃO POSSA SER EXECUTADA.**

A ula seria a responsável por realizar esta função. Bastava apenas inserir dentro da mesma um registrador de deslocamento de 8 bits (reverso, pois os registradores de deslocamente geralmente deslocam para a direita). O clock do registrador seria feito por um sinal proveniente da unidade de controle que poderíamos chamar de SLA, assim sempre que a ULA recebesse SLA ela saberia que deve pegar os valores do acumulador e deslocá-los para a esquerda.

**4. DESEJAMOS ACRESCENTAR A INSTRUÇÃO JR (JUMP RELATIVE – DESVIO DO FLUXO DE EXECUÇÃO RELATIVO AO ENDEREÇO DA PRÓPRIA INSTRUÇÃO) AO CAMINHO DE DADOS DO PROCESSADOR NEANDER. FAÇA AS INCLUSÕES E MODIFICAÇÕES NECESSÁRIAS NO CAMINHO DE DADOS PARA QUE ESTA INSTRUÇÃO POSSA SER EXECUTADA.**

Uma instrução JumpRelative teria um mnemônico do tipo JR end. O funcionamento em alto nível da instrução seria PC<-PC+end. O jumprelative pula para o end bytes após o endereço onde o fim da instrução está. Adicionaremos as seguintes conexões e elementos:

* Um multiplexador na entrada X da ULA. Ao selecionar 0 o que estiver no acumulador passará para essa entrada. Ao selecionar 1 o que estiver no PC passará para a entrada.
* Uma conexão entre a saída da ULA e o PC.
* Um segundo mux na entrada do PC para ele diferenciar os valores que vem da ULA e do RDM.

O funcionamento do NEANDER para SELJR = 0 e SELPC = 0 é normal.

Vejamos o funcionamento para o caso da instrução JR.

Suponha que a instrução JR esteja na posição 0 da memória e o endereço 10h esteja na posição 1.

1. O PC é passado para o REM com SEL = 0;
2. O PC é incrementado.
3. A memória é lida no endereço 0.
4. A instrução JR é transferida para o RDM e depois para o RI.
5. Ocorre a decodificação da instrução.
6. O PC é passado para o REM com SEL=0;
7. O PC é incrementado.
8. A memória é lida no endereço 1.
9. O valor 10h é transferido para o RDM e permanece lá.
10. O valor de SELJR é setado para 1.Isso implica que na entrada da ULA existem os valores X = PC e Y = RDM.
11. Um sinal de ADD é dado a ULA.
12. O sinal S de saída aparecerá na entrada do Mux antes do PC
13. O SELPC receberá 1 para indicar que o PC receberá o que saiu da ULA. Então PC receberá um sinal Carga PC.
14. O desvio foi completo.

5. Faça uma sugestão de implementação do processador Neander monociclo. Avalie os benefícios e custos desta implementação, comparando com a versão multiciclo.

6. Faça sugestões de como melhorar o desempenho das implementações monociclo e multiciclo dos processadores estudados.

**7. DEFINA PROCESSADOR MULTICICLO E MONOCICLO. INDIQUE DUAS VANTAGENS DE CADA UM DELES SOBRE O OUTRO.**

Processadores monociclo: Cada instrução leva um ciclo de clock para ser implementada. Certas instruções poderiam ser executadas mais rapidamente que em um processador, pois não há tempo gasto esperando a unidade mais lenta ser completada. CPI ideal de 1.

Processadores multiciclo: Cada instrução leva mais de um ciclo de clock para ser concluída. Não há desperdício de tempo de instruções mais lentas. Pode-se usar cada unidade funcional mais de uma vez na instrução.

*Modos de endereçamento*

**1. DEFINA E EXEMPLIFIQUE OS MODOS DE ENDEREÇAMENTO: IMEDIATO, DIRETO, INDIRETO E INDEXADO.**

* Modo de endereço imediato: O operando faz parte da instrução, assim não há referência a memória para buscá-lo. Contudo a faixa do operando fica limitada pelo tamanho da instrução menos o opcode. Ex: ADD 5. Adiciona o valor 5 ao acumulador.
* Modo de endereçamento direto: A instrução contém o endereço do operando. É necessário então buscar o operando naquele endereço de memória. O espaço de endereçamento é limitado pelo tamanho da instrução menos o opcode. Ex: ADD x, adiciona ao acumulador o valor no endereço x.
* Modo de endereçamento indireto: A instrução contém o endereço do endereço do operando. É preciso buscar na memória duas vezes para achar o operando. As possibilidades de endereço são maiores. Ex: Add x, adiciona ao acumulador o valor no endereço apontado por x.
* Modo de endereçamento indexado: O endereço do operando é formado pela soma de um offset e de um valor base que pode esta contido em um registrador. Ex: ADD 4(x) Soma 4 com x para obter o valor do endereço do operando que será somado ao acumulador.

**2. CLASSIFIQUE AS INSTRUÇÕES DO PROCESSADOR NEANDER QUANTO AO NÚMERO DE ENDEREÇOS.**

O Neander tem instruções de 1 endereço ou de 0 endereços.

STA, LDA, ADD, OR, AND, JMP, JN, JZ são instruções de 1 endereço. NOP, NOT e HLT são de 0 endereços.

**3. CLASSIFIQUE AS INSTRUÇÕES ABAIXO, DO PROCESSADOR MIPS, QUANTO AO NÚMERO DE ENDEREÇOS E QUANTO AO MODO DE ENDEREÇAMENTO.**

sw $5, 100($2)

Instrução de 2 endereços com endereçamento indexado.

add $3, $2, $4

Instrução de 3 endereços com endereçamento por registrador.

**4. PESQUISE UM PROCESSADOR COMERCIAL, SELECIONE DUAS INSTRUÇÕES E CLASSIFIQUE-AS QUANTO AO NÚMERO DE ENDEREÇOS E QUANTO AO MODO DE ENDEREÇAMENTO.**

Intel 8051

MOV R1,#3 – 1 Endereço, endereçamento imediato

JMP INICIO – 1 Endereço, endereçamento direto

MOVC A,@A+DPTR – Endereçamento indexado

**5. CITE E EXPLIQUE 2 TIPOS DE OPERAÇÕES/INSTRUÇÕES COMUNS EM PROCESSADORES.**

Instruções de movimentação de dados – Loads e Stores

Instruções aritiméticas – Add, sub, and, or.

**6. QUANTAS VEZES A CPU ACESSA A MEMÓRIA QUANDO BUSCA E EXECUTA UMA INSTRUÇÃO COM MODO** **DE ENDEREÇAMENTO INDIRETO, SE A INSTRUÇÃO É:**

a) uma computação que requer apenas um operando?

Busca = 1 acesso a memória para a busca da instrução.

Execução = 2 acessos a memória para a busca do operando pelo endereçamento indireto.

Total = 3 acessos a memória.

b) um desvio?

Busca = 1 acesso a memória para a busca da instrução.

Execução = 0 acesso a memória.

Total = 1 acesso a memória.

*Avaliação de desempenho*

**1. DESEJA-SE COMPARAR O DESEMPENHO DE DOIS COMPUTADORES DIFERENTES M1 E M2. AS SEGUINTES MEDIÇÕES FORAM FEITAS NESTES COMPUTADORES.**

|  |  |  |
| --- | --- | --- |
| Programa | Tempo em M1 | Tempo em M2 |
| 1 | 2 segundos | 1,5 segundos |
| 2 | 5 segundos | 10 segundos |

**a) Suponha que M1 custe R$ 500,00 e M2 custe R$800,00. Se precisasse usar o programa 1 um grande número de vezes, qual computador você compraria e por quê?**

Suponha que o programa 1 seja executado 90% do tempo e o programa 2 apenas 10%.

Tempo de M1 = 2\*0,9 + 5\*0,1 = 2,3 segundos.

Tempo de M2 = 1,5\*0,9 + 10\*0,1 = 2,35 segundos.

M1 e M2 tem um tempo de execução aproximado. M2 é um pouco mais lento para os dois programas. Contudo, como executarei o programa um muitas vezes, seria interessante ter um computador que o fizesse de forma rápida. Caso o programa 1 fosse executado menos vezes, o seu tempo de execução não seria tão importante e eu poderia comprar um computador mais lento (e mais barato). Logo eu compraria o M2.

**2. SUPONHA QUE OUTRO USUÁRIO POSSUI AS SEGUINTES NECESSIDADES: O P1 PRECISA SER EXECUTADO 1600 VEZES A CADA HORA. QUALQUER TEMPO RESTANTE É USADO PARA EXECUTAR O P2. SE O COMPUTADOR POSSUI DESEMPENHO SUFICIENTE PARA EXECUTAR O PROGRAMA 1 A QUANTIDADE NECESSÁRIA DE VEZES POR HORA, ENTÃO, O DESEMPENHO É MEDIDO PELA VAZÃO PARA O PROGRAMA. QUE COMPUTADOR É MAIS RÁPIDO PARA ESTE WORKLOAD? Que computador é mais econômico?**

Para M1:

Tempo para P1 3200 segundos

Tempo Para P2 400 segundos, 200 execuções

Utilizando porcentagem para calcular o tempo médio de execução do workload: P1 executa 2\*(3200\*1)/3600 + 5\*(400\*1)/3600 = 2,3 segundos

Para M2:

Tempo para P1 2400 segundos

Tempo para P2 1200 segundos, 120 execuções

De forma análoga, 1,5\*2400\*1/3600 + 10\*1200\*1/3600 = 4,3 segundos

Neste caso o computador M1 é mais rápido para o worload indicado.

**3. CONSIDERE DUAS IMPLEMENTAÇÕES DIFERENTES DO MESMO CONJUNTO DE INSTRUÇÕES (M1 E M2). HÁ TRÊS CLASSES DE INSTRUÇÕES (A, B E C) NO CONJUNTO DE INSTRUÇÕES. A IMPLEMENTAÇÃO M1 POSSUI UM CLOCK DE 6,0GHZ E A IMPLEMENTAÇÃO M2 UM CLOCK DE 3,0GHZ. A TABELA FORNECE UM NÚMERO MÉDIO DE CICLOS PARA CADA CLASSE DE INSTRUÇÃO EM CADA IMPLEMENTAÇÃO. ALÉM DISSO, É FORNECIDA A PROPORÇÃO MÉDIA DE CLASSES DE INSTRUÇÃO GERADA PELO COMPILADOR.**

|  |  |  |  |
| --- | --- | --- | --- |
| Classe | CPI em m1 | CPI em m2 | Uso pelo compilador |
| A | 2 | 1 | 50% |
| B | 3 | 2 | 25% |
| C | 5 | 2 | 25% |
|  |  |  |  |

Qual das duas implementações oferece melhor desempenho?

Para a implementação 1:

Ciclos de Clock = 2\*0,5 + 3\*0,25 + 4\*0,25 = 1 + 0,75 + 1 = 2,75.

Tempo de CPU = 2,75/6\*10^6 = 458ms

Para a implementação 2:

Ciclos de Clock = 0,5+2\*0,25+2\*0,25 = 1,5

Tempo de CPU = 1,5/3\*10^6 = 500ms

A implementação 1 fornece o melhor desempenho.

**4. DEFINA SOFTWARE DE BENCHMARK. CITE 2 EXEMPLOS.**

Software de benchmarks são programas que possibilitam o teste do desempenho de hardware de determinados sistemas de computador. O software executa operações e analisa o desempenho de um processador ao executar tais operações, o mesmo software pode ser executado em outra maquina para formar um paralelo comparativo entre as duas. Dhrystone, whetstone são exemplos.

*Pipeline*

1. Mostre, através de esboço gráfico, que quanto mais estágios contiver um pipeline ideal, maior será o número de instruções executadas no final de um período. Considere para tal uma sequência de 4 slots de tempo, um caso inicial sem a adoção de pipeline e pipelines de 2, 3 e 4 estágios.

**2. O PIPELINE DA QUESTÃO ANTERIOR É UM PIPELINE IDEALIZADO. CITE DOIS MOTIVOS PELOS QUAIS, NA PRÁTICA, NÃO SE OBTÉM O RESULTADO ILUSTRADO NA QUESTÃO ANTERIOR.**

Primeiramente para se obter um pipeline com ganho máximo, é necessário um fluxo infinito (sempre continuo) de instruções, ou seja, é necessário que o pipeline sempre esteja cheio. Segundo, o exemplo prevê que o número de estágios do pipeline tem tempo idêntico, além disso ele também considera que cada instrução possui o mesmo número de estágios, ambas as condições nem sempre são verdadeiras.

3. Identifique todas as dependências de dados existentes no código a seguir. Quais dependências são conflitos que podem ser resolvidos com forwarding (adiantamento de dados)?

1. add $2, $5, $4
2. add $4, $2, $5
3. sw $5, 100($2)
4. add $3, $2, $4

Dependência entre 1 e 2 por conta do registrador $2, pode ser resolvido com fowarding ULA-ULA.

Dependência entre 1 e 3 por conta do registrador $2, pode ser resolvido com fowarding do registrador após a memória para a ULA.

Dependência entre 1 e 4 por conta do registrador $2, dependência resolvida por conta do número de instruções envolvidas no entre 1 e 4.

Dependência entre 2 e 4 por conta do registrador $4, dependência resolvida com fowarding do registrador após a memória para a ULA.

**4. EXECUTE O CÓDIGO ABAIXO NO CAMINHO DE DADOS DO MIPS SEM FORWARDING (ADIANTAMENTO DE DADOS).**

**FAÇA O DIAGRAMA DE TEMPO DA PIPELINE. SE EXISTEM STALLS, EXPLIQUE O MOTIVO DE CADA UM. CALCULE O CPI PARA O PROGRAMA.**

lw $1, 0($4)

add $3, $1, 3

add $2, $3, 3

sub $3, $1, $3

sw $2, 0($4)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | INSTRUCTION | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 | t9 | t10 | t11 | t12 | t13 |
| 1 | lw $1, 0($4) | IF | REG | ULA | MEM | WB |  |  |  |  |  |  |  |  |  |
| 2 | add $3, $1, 3 |  | IF | stall | stall | REG | ULA | MEM | WB |  |  |  |  |  |  |
| 3 | add $2, $3, 3 |  |  |  |  | IF | stall | stall | REG | ULA | MEM | WB |  |  |  |
| 4 | sub $3,$1,$3 |  |  |  |  |  |  |  | IF | REG | ULA | MEM | WB |  |  |
| 5 | sw $2, 0($4) |  |  |  |  |  |  |  |  | IF | stall | REG | ULA | MEM | WB |

O stall na instrução 2 é causado pelo fato de o resultado do registrador 1 ainda não ter sido gravado, então é necessário esperar que o valor seja gravado para poder ser lido na instrução 2. Read after write hazard.

O stall na instrução 3 é causado pelo fato de o registrador 3 ainda não ter sido gravado e seu valor precisar ser utilizado nesta instrução. É necessário esperar a gravação. Read after write hazard.

O stall na instrução 5 é causado pelo fato de o valor do registrador 2 ainda não ter sido gravado quando precisa ser utilizado por esta instrução, logo é preciso esperar a gravação. Também um read after write hazard.

CPI = 14 ciclos/ 5 instruções = 2,8 ciclos por instrução.

Agora refaça o diagrama de tempo considerando o caminho de dados com forwarding.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | INSTRUCTION | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | t8 | t9 |
| 1 | lw $1, 0($4) | IF | REG | ULA | MEM | WB |  |  |  |  |  |
| 2 | add $3, $1, 3 |  | IF | REG | stall | ULA | MEM | WB |  |  |  |
| 3 | add $2, $3, 3 |  |  | IF | stall | REG | ULA | MEM | WB |  |  |
| 4 | sub $3,$1,$3 |  |  |  |  | IF | REG | ULA | MEM | WB |  |
| 5 | sw $2, 0($4) |  |  |  |  |  | IF | REG | ULA | MEM | WB |

O stall da instrução 2 foi diminuído pela introdução de um fowarding de memória para a ULA, entretanto ainda é necessário esperar que a memória leia o valor que será colocado em 1. O valor já é adiantado para a entrada da ULA para que seja somado a 3 e salvo no registrador 3.

O stall da instrução 3 foi diminuído graças a um fowarding da ULA para a ULA, entretanto a unidade de banco de registradores está ocupada duranto ciclo t3 e não é possível utilizá-la, gerando um stall.

Um novo stall apareceria para a instrução 4, mas isso é resolvido com o fowarding de ULA para ULA. O stall apareceria, pois é necessário a utilização do registrador 3 que ainda não foi salvo, mas seu resultado esta sendo executado na ULA.

Um Fowarding de ULA para Memória adianta o valor do registrador 2 gerado pela instrução 3 que será utilizado pela instrução 5.

O CPI então será = 10 ciclos/ 5 instruções = 2.

**5. DADO UM PROCESSADOR MIPS COM UM PIPELINE DE 5 ESTÁGIOS, ESCREVA UM TRECHO DE CÓDIGO EM LINGUAGEM DE MONTAGEM EM QUE OCORRAM AS SEGUINTES SITUAÇÕES:**

**A. ADIANTAMENTO DE DADOS DA SAÍDA PARA A ENTRADA DA ALU;**

add $3, $1, $2

sub $4, $3, $5

b. Adiantamento de dados da saída do estágio de memória para a entrada da ALU;

lw $1, 100($0)

add $2, $1, $3

c. Uma bolha no pipeline devido a uma dependência de controle.

Eu não sei o resultado do teste do desvio até o estágio pós ULA.

beq $1, $2, 100

add $4, $5, $6

**6. CONSIDERE O SEGUINTE TRECHO DE CÓDIGO EM LINGUAGEM DE MONTAGEM:**

add R5, R0, R0

add R20, R0, #40

Soma:

ld R10, A(R20)

add R5, R5, R10

sub R20, R20, #4

bne R20, R0, Soma

add R5, R0, R0

add R20, R0, #40

Soma:

ld R10, A(R20)

sub R20, R20, #4

Nop

add R5, R5, R10

bne R20, R0, Soma

Assuma que o pipeline do processador não possui mecanismos de “stalls” ou adiantamento de dados. Reescreva o código inserindo o menor número possível de nops para eliminar as dependências de dados. Se for possível, reordene as instruções para minimizar o número de nops. R0 sempre contém o valor 0.

**7. DESCREVA OS TIPOS DE DEPENDÊNCIAS DE DADOS, COMO E QUANDO ELAS OCORREM E QUAIS AS TÉCNICAS PARA RESOLVÊ-LAS POR HARDWARE E POR SOFTWARE?**

Dependência verdadeira dependência que ocorre quando se quer ler um dado que ainda não foi escrito corretamente. Ocorre quando duas instruções são executadas uma atrás da outra onde a segunda exige um dado que a primeira ainda não terminou de processar.

Dependência de saída, ou dependência falsa, ocorre quando duas operações de escrita são feitas ao mesmo tempo. Ou seja, uma segunda instrução que escrever um dado (em um registrador por exemplo) e a segunda instrução que escrever no mesmo local.

Antidependência, duas instruções seguidas, a primeira quer ler um dado num local que a segunda utiliza como local de escrita. A segunda instrução então não pode escrever um dado antes que a primeira leia-o.

Organizando as instruções de modo a por um delay entre uma instrução e outra que tem dependências verdadeiras é um modo de resolver dependências por software. Fowarding é o método empregado utilizando hardware.

**8. A TABELA MOSTRA O TEMPO DE ATRASO INTRODUZIDO POR CADA UMA DAS UNIDADES DO CAMINHO DE DADOS ESTUDADO NAS AULAS.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Unidade | Mem. Instr. | Banco de Reg. | Alu | Mem. Dados |
| Tempo de atraso ps | 200 | 100 | 200 | 200 |

a) Suponha que o tempo de execução da ALU é encurtado de 25%. Esta alteração afeta o desempenho do pipeline? Em caso afirmativo, determine a alteração de desempenho.

Supondo uma diminuição do tempo da ULA em 25% não influencia o desempenho do pipeline, visto que o tempo de cada estágio se baseia no estágio mais demorado, que no exemplo são os estágios que envolvem a memória.

b) Responda à questão anterior, supondo que o tempo de execução da ALU aumenta 25%.

Aumentando o tempo da ULA em 25% aumentaria o tempo de seu estágio, isso significa que os outros estágios tem que esperar pelo estágio da ULA, que seria o mais demorado. Supondo o pipeline cheio 5 instruções seriam executadas em 1000ps+4x200ps = 1800ps.

Nesta nova implementação o tempo da ula eh de 250ps o que faz deste tempo o padrão para todos os estágios. Logo, supondo o pipeline cheio teríamos o tempo para executar 5 instruções 1250ps + 2x250ps = 2250ps.

*Processadores RISC*

**1. O QUE CARACTERIZA UM MICROPROCESSADOR COM ARQUITETURA RISC? DÊ EXEMPLOS DE DISPOSITIVOS COMERCIAIS (PROCURE EXEMPLOS NA INTERNET).**

A arquitetura RISC é caracterizada por instruções mais simples e um número menor de instruções. Além disso, computadores RISC possuem maior adaptabilidade para trabalhar com pipeline e possibilitam a diminuição do trabalho da unidade de controle, diminuindo seu tamanho efetivamente. Com isso pode-se aumentar o número de registradores disponíveis no chip. Exemplos de dispositivos são o Motorola 88000 e o MIPS.

**2. O QUE CARACTERIZA UM MICROPROCESSADOR COM ARQUITETURA CISC? DÊ EXEMPLOS DE DISPOSITIVOS COMERCIAIS (PROCURE EXEMPLOS NA INTERNET).**

Como o nome diz, a arquitetura CISC possui instruções mais complexas capazes de realizar várias operações de uma só vez. Intel 386 e o Motorola 68020 são exemplos de dispositivos que usam a arquitetura CISC.

3. Vantagens e desvantagens da arquitetura RISC.

**4. É FÁCIL IDENTIFICAR CLARAMENTE SE UM PROCESSADOR SEGUE UMA ARQUITETURA RISC OU CISC?**

**POR QUE?**

Atualmente as implementações de diversas arquiteturas tornaram-se um misto de RISC e CISC. Arquiteturas que se dizem CISC são na verdade “interfaceadas” com instruções RISC enquanto que arquiteturas RISC aumentaram a complexidade e o número de suas instruções de modo que tornaram-nas muito parecidas com arquiteturas CISC.

Leitura: RISC vs. CISC: the Post-RISC Era (http://arstechnica.com/cpu/4q99/risc-cisc/rvc-1.html)