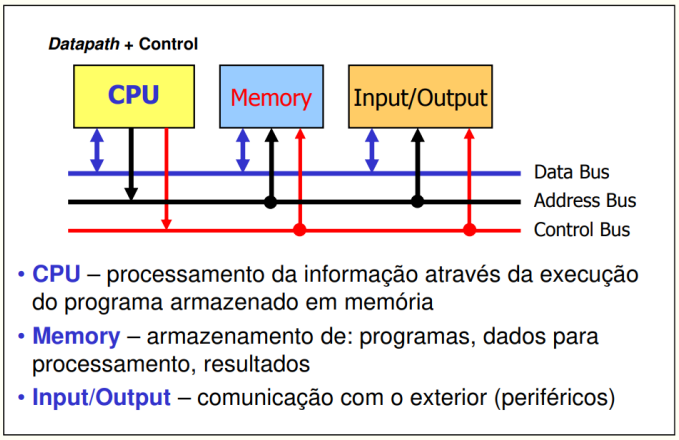
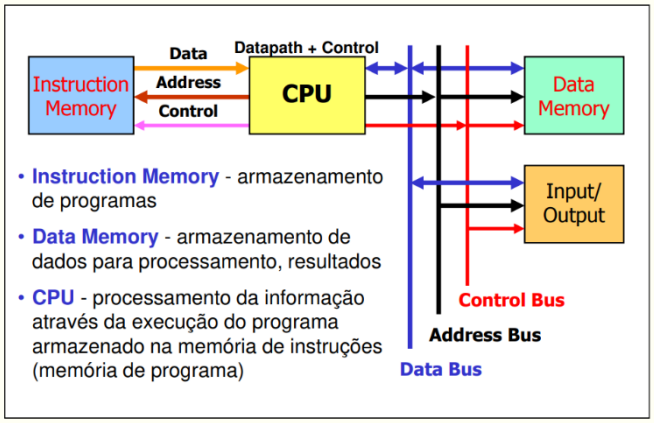
Modelo de von Neumann



- **CPU** – processamento da informação através da execução do programa armazenado em memória
- **Memory** – armazenamento de: programas, dados para processamento, resultados
- **Input/Output** – comunicação com o exterior (periféricos)

Modelo de Harvard



- **Instruction Memory** - armazenamento de programas
- **Data Memory** - armazenamento de dados para processamento, resultados
- **CPU** - processamento da informação através da execução do programa armazenado na memória de instruções (memória de programa)

von Neumann *versus* Harvard – resumo

- **Modelo de von Neumann**
  - um único espaço de endereçamento para instruções e dados (i.e. uma única memória)
  - acesso a instruções e dados é feito em ciclos de relógio distintos
- **Modelo de Harvard**
  - dois espaços de endereçamento separados: um para dados e outro para instruções (i.e. duas memórias independentes)
  - possibilidade de acesso, no mesmo ciclo de relógio, a dados e instruções (i.e. CPU pode fazer o *fetch* da instrução e ler os dados que a instrução vai manipular no mesmo ciclo de relógio)
  - memórias de dados e instruções podem ter comprimentos de palavra diferentes

Arquitetura básica do CPU

- Secção de dados (datapath) – elementos operativos/funcionais para encaminhamento, processamento e armazenamento de informação (Mux, ,ALU, Registos Internos, etc)
- Unidade de Controlo – responsável pela coordenação dos elementos do datapath durante a execução de um programa

O CPU é sempre uma máquina de estados síncrona

Ciclo-base de execução de uma instrução

- Instruction fetch : leitura do código máquina da instrução (instrução reside em memória)
- Instruction decode : descodificação da instrução pela unidade de controlo
- Operand fetch : leitura do(s) operando(s)
- Execute : execução da operação especificada pela instrução
- Store result : armazenamento do resultado da operação no destino especificado na instrução

Tipos de Arquiteturas

- Arquiteturas Register-Memory : Operandos das intruções aritméticas e lógicas residem em registos internos do CPU ou em memória
- Arquiteturas Load-Store : Operandos das instruções aritméticas e lógicas residem em registos internos do CPU de uso geral (mas nunca na memória)

Aspetos Chave da Arquitetura MIPS

- 32 Registos de uso geral, de 32 bits cada (1 word ← → 32 bits)
- ISA baseado em instruções de dimensão fixa (32 bits)
- Arquitetura load-store (register-register operantion)
- Memória organizada em bytes (byte addressable)
- Espaço de endereçamento de 32 bits (2^{32} endereços possíveis, máximo de 4GB de memória)
- Barramento de dados externo de 32 bits

Formatos de Instruções no MIPS

6 Campos : opcode (6b), rs (5b), rt (5b), rd (5b), shamt (5b) e funct (6b)

4 Campos: opcode (6b), rs (5b), rt (5b), offset (16b)

2 Campos : opcode (6b), endereço (26b)

- Formato R (opcode = 0) : instruções aritméticas e de desvio, adição, subtração e branch.
- Formato I : instruções de memória, carregar e armazenar, instruções aritméticas com um operando imediato. Tem 4 Campos
- Formato J : instruções de desvio incondicional, como jump. Tem 2 Campos

Decomposição de Instruções Virtuais:

- move Rdst, Rsrc → or Rdst, Rsrc, $0
- bge Rsrc1, Rsrc2, Label → slt Rdst, Rsrc1, Rsrc2

  beq Rsrc1, Rsrc2, Label
- bgt Rsrc1, Rsrc2, Label → slt Rdst, Rsrc1, Rsrc2

  bne Rsrc1, Rsrc2, Label

Acesso a informação residente na memória externa:

Um registo é capaz de conter o endereço de memória a aceder (no MIPS um registo interno permite armazenar 32 bits). Endereçamento indireto por registo

Solução: Código máquina da instrução:

- Opcode (Load/Store)(6b), R1 (5b), R2 (5b), offset (16b)

Leitura e Escrita na Memória:

- LW – transfere uma palavra de 32 bits da memória para um registo interno do CPU (1 word é armazenada em 4 posições de memória consecutivas)
- SW – transfere uma palavra de 32 bits de um registo interno do CPU para a memória (1 word é armazenada em 4 posições de memória consecutivas)

O acesso a words só é possível em endereços múltiplos de 4

Organização das words de 32 bits na memória:

Ordem de armazenamento dos bytes na memória:

- Big-endian : byte mais significativo armazenado no endereço mais baixo da memória
- Little-endian : byte menos significativo armazenado no endereço mais baixo da memória

O MARS usa a implementação Little-endian

Codificação de Branches

- Endereço-alvo = PC (Program Counter) + offset
- Offset pode ser negativo ou positivo
- O PC é incrementado na fase “fetch” da instrução (Instruction Fetch)

Uma vez que um registo é capaz de armazenar 32 bits, este pode ser usado para endereçamento indireto por registo, passando assim o salto a ser feito através do mesmo. Para isto é usada a instrução jr (jump register), em vez da instrução j (jump)

Codificação de Instruções que usam constantes

- opcode (6b), RS (5b), RT (5b), Imm (16)

Modos de Endereçamento no MIPS

- Instruções aritméticas e lógicas : endereçamento tipo registo
- Instruções aritméticas e lógicas com constantes : endereçamento imediato
- Instruções de acesso à memória : endereçamento indireto por registo com deslocamento
- Instruções de salto condicional (branches) : endereçamento relatico ao PC
- Instrução de salto incondicional através de um registo (instrução JR) : endereçamento indireto por registo
- Instrução de salto incondicional (J) : endereçamento direto

Sub-Rotinas

- Estratégicas para salvaguardar registos:
- Estratégia “caller-saved” : o programa “chamador” salvaguarda o conteúdo dos registos, sendo que depois repõe o conteúdo dos mesmos
- Estrategia “calle-saved” : a sub-rotina salvaguarda o conteúdo dos registos, sendo que depois repõe o conteúdo dos mesmos
- A salvaguarda dos registos é feita na stack, através do $sp (stack pointer)
- A stack funciona através da estratégia LIFO (Last In First Out)

Overflow :

- Com sinal: Carry-in XOR Carry-out
- Sem sinal: MSBop1 XOR MSBres

Multiplicação de quantidades

- Com Sinal : Algoritmo de Booth
- Sem Sinal : decimal clássico

[Rsrc1] x [Rsrc2] = [HI][LO]

- O registo HI armazena os 32 bits mais significativos do resultado
- O registo LO armazena os 32 bits menos significativos do resultado

Para mover os valores destes registos : mfhi Rdst, mflo Rdst

Numa multiplicação da qual resulte um número de 32 bits:

- Caso o MSB de LO seja 1 → HI = 0xFFFF
- Caso o MSB de LO seja 0 → HI = 0x0000

Divisão de Inteiros

Continua a existir a necessidade de 64 bits para armazenar o resultado final na forma de um quociente e de um resto

- O registo HI armazena o resto da divisão inteira
- O registo LO armazena o quociente da divisão inteira

[Rsrc1] : [Rscr2] = [HI][LO]

- Rsrc1 é o dividendo, e o Rsrc2 é o divisor

Representação de quantidades fracionárias:

- O número de bits da parte inteira determina a gama de valores representáveis.
- O número de bits da parte fracionária determina a precisão da representação

Precisão Simples (32b):

- S (1b), Expoente (8b), Mantissa (23b)
- Expoente [-126, +127]
- Codigos [1 a 254]
- Caso E = 0 : não representável
- Overflow (E > 254) → infinito (divisão por 0), gama excedida
- Underflow (E < 1) → infinito
- Nan : 0.0 : 0.0, inf : inf, nan * 2

Precisão Dupla (64b):

- S (1b), Expoente (11b), Mantissa (52b)
- Expoente [-1022, +1023]
- Codigos [1 a 2046]
- Overflow (E > 2046) → infinito
- Underflow (E < 1) → não representável
- Expoente : Exp = E – 127
- Mantissa : 1 + parte fracionária

Precisão Simples		Precisão Dupla		Representa
Expoente	Parte Frac.	Expoente	Parte Frac.	
0	0	0	0	0
0	≠ 0	0	≠ 0	Quantidade desnormalizada
1 a 254	*qualquer*	*1 a 2046*	*qualquer*	*Nº em vírgula flutuante normalizado*
255	0	2047	0	Infinito
255	≠ 0	2047	≠ 0	NaN (Not a Number)

Datapath

- Secção de Dados (datapath) – Registos, ALU, Mux
- Unidade de Controlo – responsável pela coordenação dos elementos da secção de dados, durante a execução de cada instrução
- Elementos Combinatórios – ALU, etc
- Elementos de estado – têm capacidade de armazenamento (registos agrupados, registos internos), possui 2 entradas, uma para dados e outra para o clock (sincrono), pode ser lido em qualquer momento. Pode ter sinais de controlo adicionais, um sinal de leitura (read) (resultado assincrono), e outro sinal de escrita (write) (sincrono)

O processo de Instruction Fetch deverá, uma vez concluído, deixar o conteúdo do PC (Program Counter) pronto para endereçar a próxima instrução

Datapath : instruções tipo R

1. Instruction Fetch : leitura da instrução, cálculo de PC+4
2. Leitura dos registos operandos : registos nos campos “rs” e “rt”
3. Realização da operação na ALU : especificada no campo “funct”
4. Escrita do resultado no registo destino : especificado no campo “rd”

Datapath : instrução SW (tipo I)

1. Instruction Fetch : leitura da instrução, cálculo de PC+4
2. Leitura do registo que contém o endereço-base e o valor a transferir
3. Realização da operação na ALU : soma algébrica entre o registo “rs” e o offset
4. Escrita na memória

Datapath : instrução LW (tipo I)

1. Instruction Fetch : leitura da instrução, cálculo de PC+4
2. Leitura do registo que contém o endereço-base
3. Realização da operação na ALU : soma algébrica entre o registo “rs” e o offset
4. Leitura da Memória
5. Escrita do valor lido da memória no registo destino : registo especificado em “rt”

Datapath : instrução de branch

1. Instruction Fetch : leitura da instrução, cálculo de PC+4
2. Leitura de dois registos
3. Comparação dos conteúdos dos registos : operação na ALU (através de slt)
4. Cálculo do endereço-alvo da instrução de branch (BTA = (PC+4)+instruction_offset << 2)
5. Alteração do valor do registo PC
    - - Se a condição testada pelo branch for verdadeira : PC = BTA
    - - Se a condição testada pelo branch for falsa : PC = PC + 4

Single-Cycle

- Unidade de controlo deve gerar os sinais, e o opcode para a ALU
- A frequência máxima do clock está limitada pelo tempo de execução da instrução "mais longa"
- O tempo de execução de uma instrução corresponde ao somatório dos atrasos introduzidos por cada um dos elementos operativos envolvidos na sua execução

Tempo de execução das instruções

- Consideremos os seguintes tempos de atraso introduzidos por cada um dos elementos operativos do *datapath single-cycle*:
  - Acesso à memória para leitura - t_{RM}
  - Acesso à memória para preparar a escrita - t_{WM}
  - Acesso ao *register file* para leitura - t_{RRF}
  - Acesso ao *register file* para preparar a escrita - t_{WRF}
  - Operação da ALU - t_{ALU}
  - Operação de um somador - t_{ADD}
  - Unidade de controlo - t_{CNTL}
  - Extensor de sinal - t_{SE}
  - Shift Left 2 - t_{SL2}
  - Tempo de *setup* do PC - t_{stPC}

- Considerando os tempos de atraso anteriores, os tempos de execução das várias instruções suportadas pelo datapath single cycle serão:

Instruções tipo R:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}) + t_{ALU} + t_{WRF}$

Instrução SW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{WM}$

Instrução LW:

- $t_{EXEC} = t_{RM} + \max(t_{RRF}, t_{CNTL}, t_{SE}) + t_{ALU} + t_{RM} + t_{WRF}$

Instrução BEQ:

- $t_{EXEC} = t_{RM} + \max(\underbrace{\max(t_{RRF}, t_{CNTL}) + t_{ALU}}_{\text{comparação}}, \underbrace{t_{SE} + t_{SL2} + t_{ADD}}_{\text{cálculo do BTA}}) + t_{stPC}$

Instrução J:

- $t_{EXEC} = t_{RM} + \max(t_{CNTL}, t_{SL2}) + t_{stPC}$

O *datapath Multi-cycle* – fases de execução

Fase 1 (memória, ALU):
- *Instruction fetch* **e** cálculo de PC+4

Fase 2 (*register file*, ALU, unidade de controlo):
- *Operand fetch* **e** cálculo do *branch target address* **e** *Instruction decode*

Fase 3 (ALU):
- Execução da operação na ALU (instruções tipo R / addi / slti), **ou**
- Cálculo do endereço de memória (instr. de acesso à memória), **ou**
- Comparação dos operandos - instrução *branch* (conclusão da instrução)

Fase 4 (memória, *register file*):
- Acesso à memória para leitura (instrução LW), **ou**
- Acesso à memória para escrita (conclusão da instrução SW), **ou**
- Escrita no *Register File* (conclusão das instruções tipo R / addi / slti: ***write-back***)

Fase 5 (*register file*):
- Escrita no *Register File* (conclusão da instrução LW: ***write-back***)

Multi-Cycle

Datapath : instruções tipo R

1. Fase 1 : instruction fetch, cálculo PC+4
2. Fase 2 : leitura dos registos, descodificação da instrução, cálculo do BTA
3. Fase 3 : cálculo da operação na ALU
4. Fase 4 : write-back

Datapath : instrução LW (tipo I)

1. Fase 1 : instruction fetch, cálculo PC+4
2. Fase 2 : leitura dos registos, descodificação da instrução, cálculo do BTA
3. Fase 3 : cálculo na ALU do endereço a aceder na memória
4. Fase 4 : leitura da memória
5. Fase 5 : write-back

Datapath : instrução BEQ (tipo R)

1. Fase 1 : instruction fetch, cálculo PC+4
2. Fase 2 : leitura dos registos, descodificação da instrução, cálculo do BTA
3. Fase 3 : comparação dos dois registos na ALU (subtração), conclusão da instrução branch com eventual escrita do registo PC com o BTA

Datapath : instrução J

1. Fase 1 : instruction fetch, cálculo PC+4
2. Fase 2 : leitura dos registos, descodificação da instrução, cálculo do BTA
3. Fase 3 : conclusão da instrução J com a seleção do JTA como próximo endereço do PC

Pipelining

- É uma técnica com algum grau de sobreposição temporal, o objetivo é aproveitar os recursos disponibilizados pelo datapath para maximizar a eficiência global do processador
- O resultado é o aumento do número de instruções processadas por unidade de tempo (throughput), isto é conseguido através de um número de tarefas executadas em paralelo

Fases :

1. Instruction fetch [IF] – ler a instrução da memória, incremento do PC
2. Operando fetch [ID] – ler os registos e descodificar a instrução
3. Execute [EX] – executar a operação ou calcular um endereço
4. Memory access [MEM] – aceder à memória de dados para leitura ou escrita
5. Write-Back [WB] – escrever o resultado no registo destino

- O instruction set do MIPS : instruções de comprimento fixo, poucos formatos de instrução, referências à memória só aparece em instruções de load/store, operandos em memória têm de estar alinhados
- Numa implementação pipeline, a unidade de controlo é uma unidade combinatória

Hazards

Tipos de hazards :

- Hazards estruturais : quando mais do que uma instrução necessita de aceder ao mesmo hardware
- Hazards de controlo : quando é necessário fazer o instruction fetch de uma nova instrução e existe numa etapa mais avançada do pipeline uma instrução que pode alterar o fluxo da execução e que ainda não terminou. Uma solução é o stalling (o progresso é parado), unidade de controlo atrasa a entrada no pipeline. Outra solução é a previsão estática “not taken”. Outra solução é o delayed branch
- Hazards de dados : resulta da dependência entre o resultado calculado por uma instrução e o operando usado por outra que segue mais atrás no pipeline. Uma solução é o forwarding (bypassing), ou seja, o resultado é disponibilizado antes de uma instrução terminar, ou seja o hazard é passado para uma instrução mais atrás no pipeline