Tempo de execução das instruções (*single-cycle*)

- A **frequência máxima** do relógio de sincronização do *datapath single-cycle* 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**
- Note-se que apenas os elementos operativos que se encontram em **série** contribuem para aumentar o tempo necessário para concluir a execução da instrução (caminho crítico)

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}

Limitações das soluções *single-cycle* - conclusões

- Num *datapath* que suporte instruções com complexidade variável, é a **instrução mais lenta que determina a máxima frequência de trabalho**, mesmo que seja uma instrução pouco frequente
- Uma vez que o ciclo de relógio é igual ao maior tempo de atraso de todas as instruções, não é útil usar técnicas que reduzam o atraso do caso mais comum mas que não melhorem o maior tempo de atraso
  - Isto contraria um dos princípios-chave de desenho: *make the common case fast* (o que é mais comum deve ser mais rápido)
- Elementos operativos que estejam envolvidos na execução de uma instrução **não podem ser usados para mais do que uma operação por ciclo de relógio** (ex: memória de instruções e de dados, ALU e somadores, ...)

O *datapath Multi-cycle*

- Uma solução para os problemas enumerados passa por abdicar do princípio de que todas as instruções devem ser executadas num único ciclo de relógio
- Em alternativa, as várias instruções que compõem o *set* de instruções podem ser executadas em vários ciclos de relógio (*multi-cycle)*:
  - A execução da instrução é decomposta num conjunto de operações
  - Em cada ciclo de relógio poderão ser realizadas **várias operações, desde que sejam independentes** (por exemplo, *instruction fetch* e cálculo de PC+4 ou *operand fetch* e cálculo do BTA)
  - Cada uma dessas operações faz uso de um elemento operativo fundamental: memória, *register file* ou ALU
- Desta forma, o período de relógio fica apenas limitado pelo maior dos tempos de atraso de cada um dos elementos operativos fundamentais
- Para os tempos de atraso que considerámos anteriormente, a máxima frequência de relógio seria assim: **fmax = 1 / t_{RM} = 1 / 5ns = 200MHz**

O *datapath Multi-cycle*

- A arquitetura *multi-cycle* do MIPS que vamos analisar adota um ciclo de instrução composto por um **máximo de cinco passos distintos**, cada um deles executado em 1 ciclo de relógio
- A distribuição das operações por estes 5 passos tenta distribuir equitativamente o trabalho a realizar em cada ciclo
- Na definição destes passos pressupõe-se que durante um ciclo de relógio apenas é possível efetuar uma das seguintes ações fundamentais da execução de uma instrução:
  - Acesso à memória externa (uma leitura ou uma escrita)
  - Acesso ao *Register File* (uma leitura ou uma escrita)
  - Operação na ALU
- No **mesmo ciclo de relógio**, podem ser realizadas operações em elementos operativos distintos, desde que sejam independentes
  - Exemplos: **um acesso à memória externa e uma operação na ALU**, ou um **acesso ao *Register File* e uma operação na ALU**

Alternativa às soluções *single-cycle*

- Uma outra vantagem duma solução de execução em vários ciclos de relógio (*multi-cycle*) é que um **mesmo elemento operativo** pode ser utilizado mais do que uma vez, no contexto da execução de uma mesma instrução, desde que em ciclos de relógio distintos:
  - A memória externa poderá ser partilhada por instruções e dados
  - A mesma ALU poderá ser usada, para além das operações que já realizava na implementação *single-cycle*, para:
    - Calcular o valor de PC+4
    - Calcular o endereço alvo das instruções de salto condicional (BTA)
- A versão ***multi-cycle*** passará assim a ter:
  - **Uma única memória** para programa e dados (arquitetura Von Neumann)
  - **Uma única ALU**, em vez de uma ALU e dois somadores

O *datapath Multi-cycle* – fases de execução

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

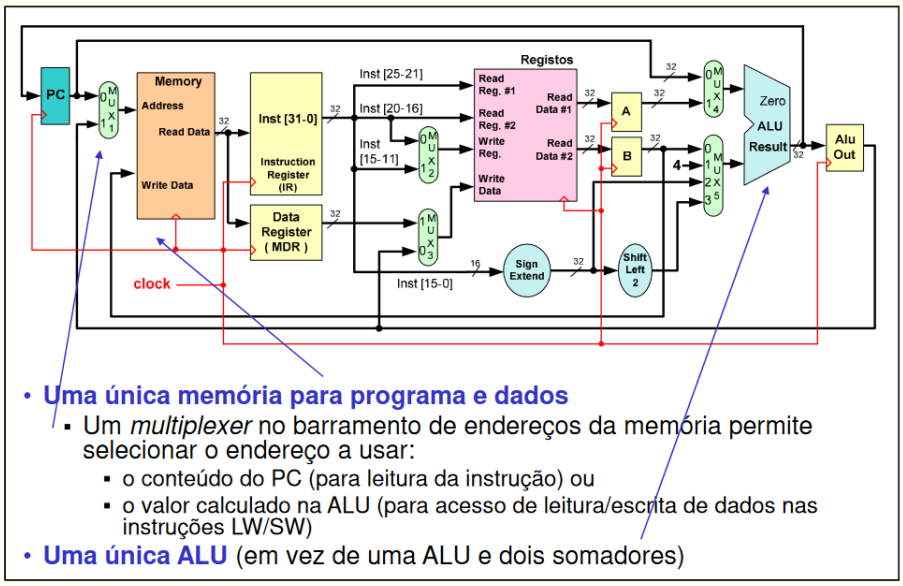
Fase 2 (*register file*, ALU, unidade de controlo):
- *Operand fetch* **e** cálculo do *branch target address* **e** *Instruction decode*

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***)

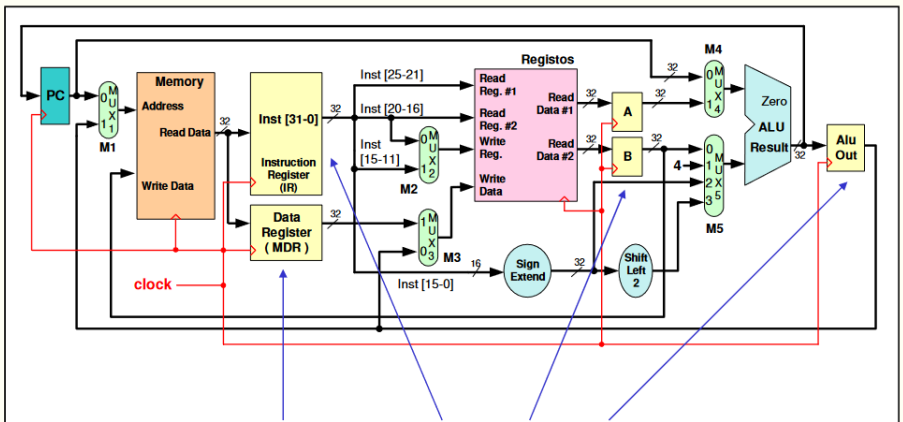
Fase 5 *(register file):*
- Escrita no *Register File* (conclusão da instrução LW: ***write-back***)

O *datapath Multi-cycle* (sem BEQ e J)



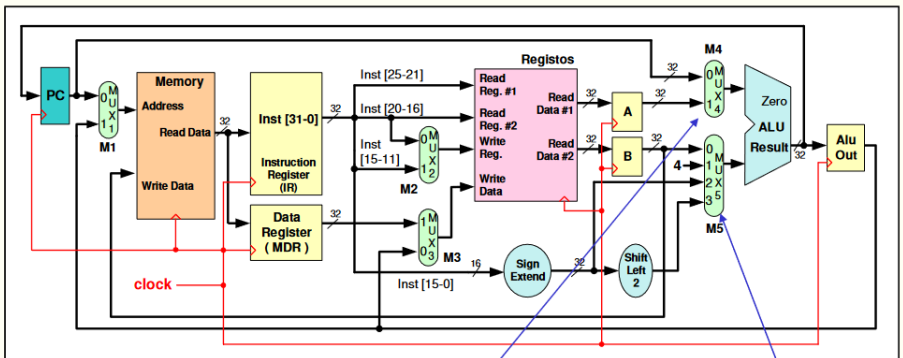
- **Uma única memória para programa e dados**
  - Um *multiplexer* no barramento de endereços da memória permite selecionar o endereço a usar:
    - o conteúdo do PC (para leitura da instrução) ou
    - o valor calculado na ALU (para acesso de leitura/escrita de dados nas instruções LW/SW)
- **Uma única ALU** (em vez de uma ALU e dois somadores)

O *datapath Multi-cycle* (sem BEQ e J)



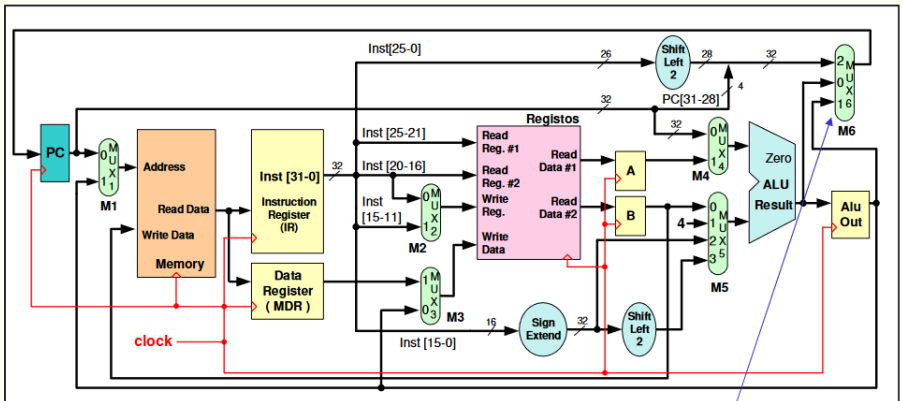
- Registos adicionados à saída dos elementos operativos fundamentais para armazenamento da informação obtida/calculada durante o ciclo de relógio corrente e que será utilizada no ciclo de relógio seguinte

O *datapath Multi-cycle* (sem BEQ e J)



- A utilização de uma única ALU obriga às seguintes alterações nas suas entradas:
  - Um *multiplexer* adicional na primeira entrada, que encaminha a saída do registo A ou a saída do registo PC
  - O *multiplexer* da segunda entrada é aumentado para poder suportar o incremento do PC (constante 4) e o cálculo do endereço alvo das instruções de branch (BTA - *branch target address*)

O *datapath Multi-cycle* com as instruções BEQ e J

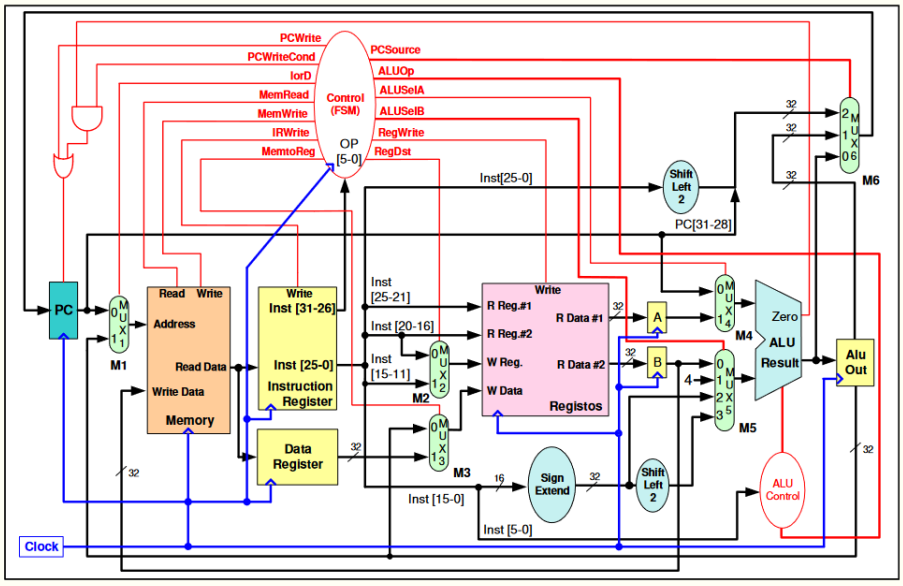


- Com as instruções de salto, o **registo PC** pode ser atualizado com um dos valores:
  - A **saída da ALU** que contém o PC+4 calculado durante o *instruction fetch* (na 1ª fase)
  - A saída do registo **ALUOut** que armazena o endereço alvo das instruções de *branch* (BTA) calculado na ALU (na 2ª fase)
  - ***Jump Target Address*** - 26 LSB da instrução multiplicados por 4 (*shift left* 2) concatenados com os 4 MSB do PC atual (o PC foi já incrementado na 1ª fase)

O *datapath Multi-cycle* – sinais de controlo

Sinal	Valor	Efeito
ALUSelB	00	A segunda entrada da ALU provém do registo indicado pelo campo rt
	01	A segunda entrada da ALU é a constante 4
	10	A segunda entrada da ALU é a versão de sinal estendido dos 16 bits menos significativos do IR (instruction register)
	11	A segunda entrada da ALU é a versão de sinal estendido e deslocada de dois bits, dos 16 bits menos significativos do IR (instruction register)
ALUOp	00	ALU efetua uma adição
	01	ALU efetua uma subtração
	10	O campo "funct" da instrução determina qual a operação da ALU
	11	ALU efetua um SLT
PCSource	00	O valor do PC é atualizado com o resultado da ALU (IF)
	01	O valor do PC é atualizado com o resultado da AluOut (Branch)
	10	O valor do PC é atualizado com o valor target do Jump
	11	Não usado

O *datapath Multi-cycle* completo



Funcionamento do *datapath* nas instruções do tipo R

- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch*
- Fase 3:
  - Cálculo da operação na ALU
- Fase 4:
  - *Write-back*

Exemplo: *add $5,$8,$6*

Funcionamento do *datapath* na instrução LW

- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch*
- Fase 3:
  - Cálculo na ALU do endereço a aceder na memória
- Fase 4:
  - Leitura da memória
- Fase 5:
  - *Write-back*

Exemplo*: lw $3,0x0014($6)*

Funcionamento do *datapath* na instrução BEQ

- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch* (BTA)
- Fase 3:
  - Comparação dos dois registos na ALU (subtração)
  - Conclusão da instrução de *branch* com eventual escrita do registo PC com o BTA

Exemplo: *beq $3,$6,endif*

Funcionamento do *datapath* na instrução J

- Fase 1:
  - *Instruction fetch*
  - Cálculo de PC+4
- Fase 2:
  - Leitura dos registos
  - Descodificação da instrução
  - Cálculo do endereço-alvo das instruções de *branch*
- Fase 3:
  - Conclusão da instrução J com a seleção do JTA como próximo endereço do PC

Exemplo: *j loop*

A unidade de controlo do *datapath Multi-cycle*

- No datapath ***single-cycle***, cada instrução é executada num único ciclo de relógio:
  - a unidade de controlo é responsável pela geração de um conjunto de sinais que não se alteram durante a execução de cada instrução.
  - a relação entre os sinais de controlo e o código de operação pode assim ser gerado por um circuito meramente combinatório.
- No datapath ***multi-cycle***, cada instrução é decomposta num conjunto de ciclos de execução, correspondendo cada um destes a um período de relógio distinto:
  - os sinais de controlo diferem de ciclo de relógio para ciclo de relógio e após o segundo ciclo diferem de instrução para instrução.
  - a solução combinatória deixa portanto de poder ser utilizada neste caso, sendo necessário recorrer a uma máquina de estados.