Arquitectura de Computadores II

LESI - 3º Ano

Revisão e implementação do *datapath* do MIPS

João Luís Ferreira Sobral
Departamento do Informática
Universidade do Minho

Janeiro 2002

Revisão do *datapath* (DP) do MIPS

- Visão global dos passos necessários para a execução de uma instrução (*lw*, *sw*, *add*, *sub*, *and*, *or*, *slt*, *beq* e *j*):

  1. busca da instrução à memória, utilizando o conteúdo do PC para indicar a posição de memória que contém a instrução a executar.

  2. descodificação da instrução e leitura do conteúdo de um ou dois registos, utilizando os campos da instrução.

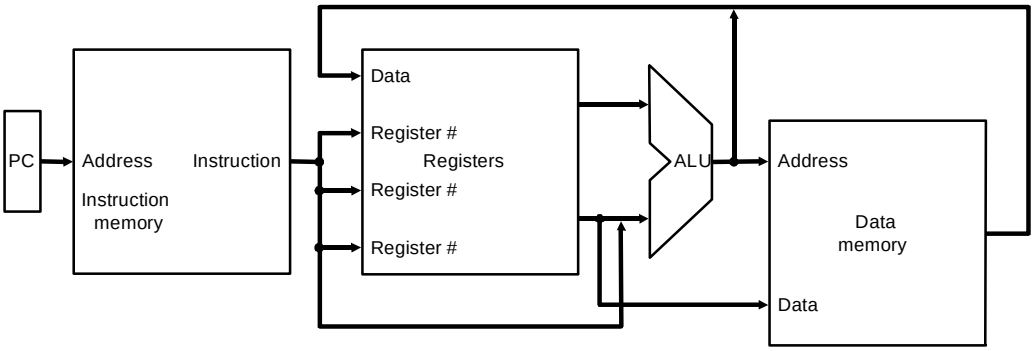
  3. execução da operação na ALU:

     a. *lw* e *sw* – calcula o endereço de memória a aceder ($rs+Imm16)
     b. *add*, *sub*, *and*, *or* e *stl* – executa a operação correspondente
     c. *beq* – efectua a comparação dos dois registos

  4. acesso à memória (*lw* e *sw*) ou escrita do resultado (*add*, *sub*, *and*, *or*, *slt* e *beq*)

  5. escrita do resultado em registos (*lw*)

- Visão global (simplificada) do DP do MIPS:



  - Os elementos que operam em dados (i.e., ALU) são de lógica combinatória.

  - Os elementos que guardam estado definem o estado da máquina (i.e., memória de instruções, de dados e banco de registos) são formados por lógica sequencial.

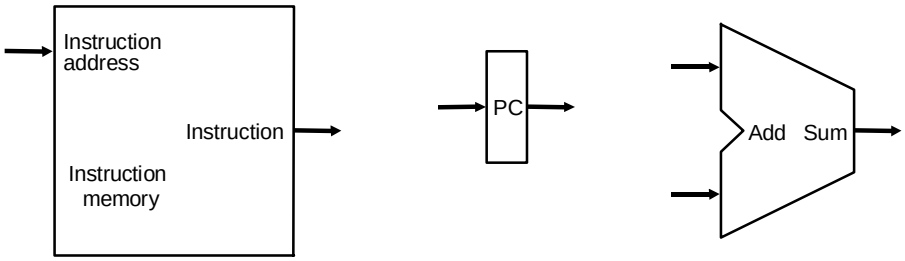
  - Os elementos que guardam estado transitam para novos valores apenas num dos bordos do relógio.

  - Nesta implementação todas as instruções são executadas num só ciclo do relógio.

 © João Luís Sobral 2002

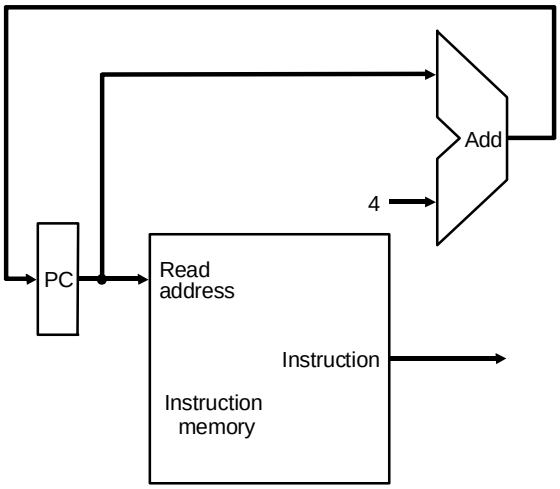
Revisão do *datapath* (DP) do MIPS

- Blocos básicos do DP - **Busca da instrução**



a. Instruction memory b. Program counter c. Adder

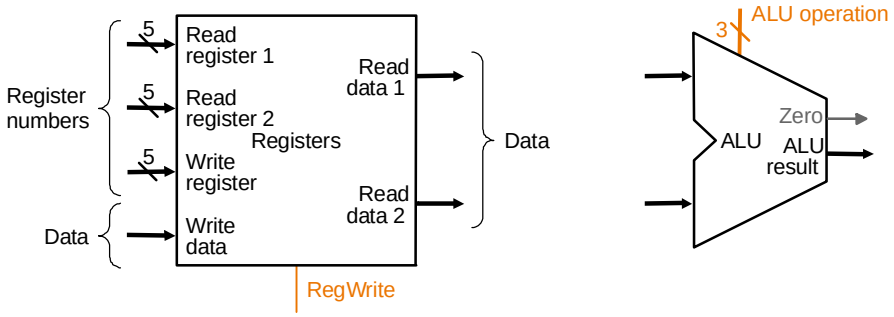
- **memória de instruções** - armazena o programa a executar
- ***program counter*** (**PC**) - indica o endereço da instrução a executar.
- **somador** - ALU que efectua apenas a operação de adição (***adder***) para calcular o endereço da próxima instrução a executar (PC + 4)



- o endereço do PC é utilizado para buscar na memória a instrução a realizar. Simultaneamente, o valor do PC é incrementado de 4 para apontar para a próxima instrução (cada instrução tem sempre 32 bits)

Revisão do *datapath* (DP) do MIPS

- Blocos básicos do DP – **Leitura do banco de registos e execução**



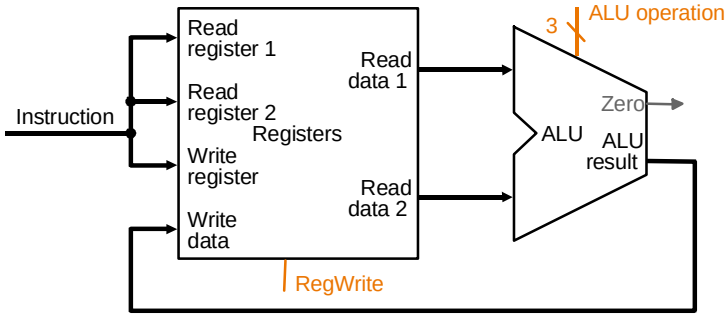
a. Registers b. ALU

- **banco de registos** - possibilita o acesso ao valor dos registos do processador. Numa instrução pode ser necessário, no máximo, ler dois registos e escrever um resultado num registo (ex. *add $rd, $rs1, $rs2*)

- O banco de registos, em cada ciclo, coloca o valor dos registos indicados em *Read register 1* e *Read register 2* em *Read data 1* e *Read data 2*, respectivamente, tornando desnecessário um sinal de controlo para a leitura dos registos.

- O quando é activado o sinal de controlo *RegWrite* o valor em *Write Data* é escrito no registo indicado em *Write register*.

- **ALU** - realiza as operações (adição, subtracção, etc..) possuindo duas entradas de dados de 32 bits, uma saída de 32 bits para o resultado e um sinal adicional que indica se o resultado da operação foi *zero*. O sinal de controlo *ALU operation* permite seleccionar a operação a realizar.



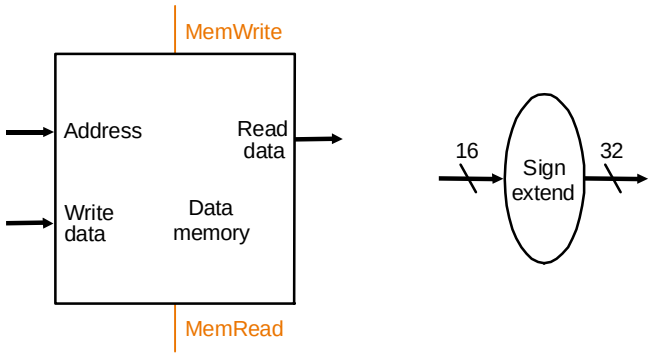
- As entradas do banco de registos (*Read register 1*, *Read register 2* e *Write register*) provêm directamente dos campos da instrução.

- As duas entradas da ALU estão ligadas às duas saídas do banco de registos.

- A saída da ALU deve ser ligada ao *Write data* do banco de registos, para que seja possível escrever o resultado da operação no registo destino (ex. add $rd, $rs1, $rs2)

Revisão do *datapath* (DP) do MIPS

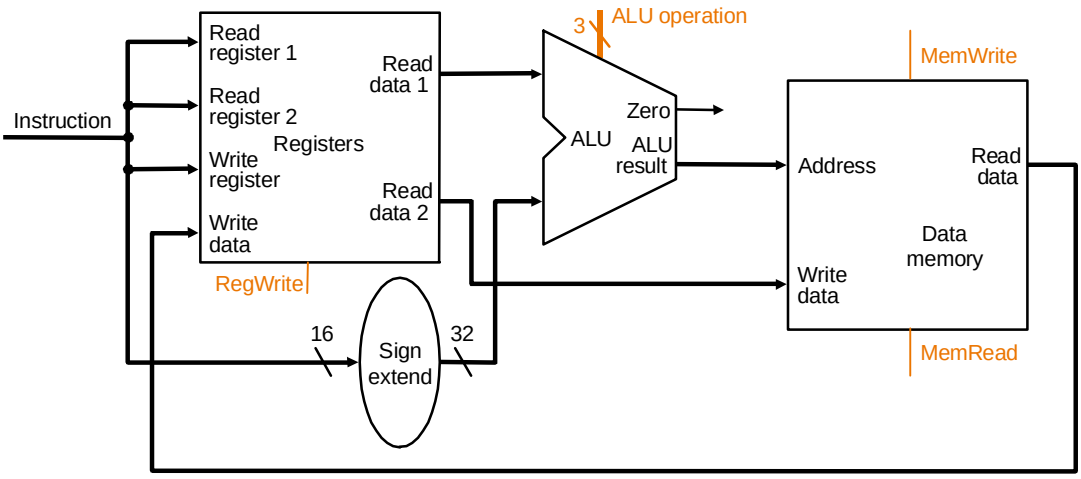
- Blocos básicos do DP – **Acessos à memória (*lw* e *sw*)**



a. Data memory unit b. Sign-extension unit

- A **unidade de memória de dados** permite ler e escrever na memória. O sinal *Address* indica a posição de memória a ler ou escrever e o sinal *Write data* indica os dados a escrever. O sinal *Read data* contém o valor lido da memória. Os sinais de controlo *MemRead* e *MemWrite* permitem seleccionar a operação de leitura ou escrita, respectivamente.

- A **unidade de extensão de sinal** efectua a extensão de valores de 16 bits, em complemento para 2, para valores de 32 bits.



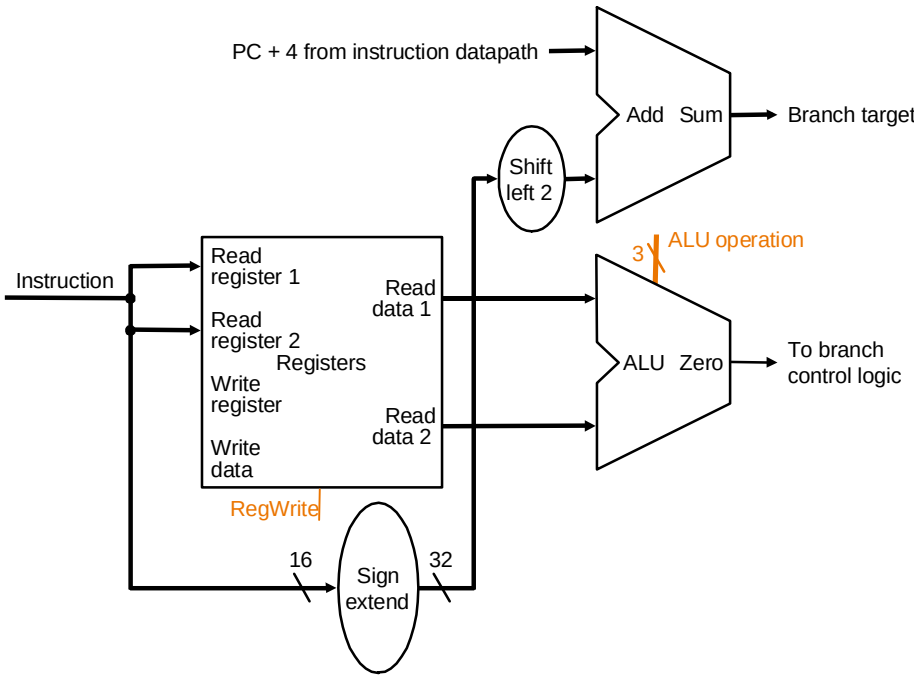
- Nos acessos à memória (i.e. *lw $rd, Imm16($rs)* e *sw $rs2, Imm16($rs1)*) o endereço a aceder é calculado através da adição de uma constante de 16 bits com o valor de um registo. Os 16 bits provém de um campo da instrução, sendo necessário estender o valor para 32 bits e ligar à ALU, bem como a primeira saída do banco de registos.

- A saída da ALU é utilizada para indicar o endereço a ser acedido

- A saída da memória é ligada ao banco de registos para possibilitar a colocação do valor lido da memória em registo (em *lw*)

Revisão do *datapath* (DP) do MIPS

- Blocos básicos do DP – **Cálculo de endereço de salto (*beq*)**



- As instruções de salto relativo (*beq $rs1, $rs2, offset*) contêm três parâmetros: os dois registos a comparar e um deslocamento de 16 bits, utilizado para calcular o endereço da próxima instrução, caso a condição se verifique.

- O endereço de salto é calculado estendendo a constante de 16 para 32 bits, multiplicando por 4 (os saltos relativos são ao nível das palavras de 4 bytes) e adicionando a PC + 4 (o salto com deslocamento 0 corresponde à instrução seguinte).

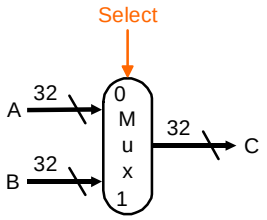
- A condição de salto é verificada subtraindo os valores dos dois registos. Quando o valor resultante é 0 é activada a saída *Zero*, que indica que o salto deve ser tomado

- Quando o salto é tomado o endereço da próxima instrução a executar é o endereço de salto calculado. Caso contrário, o salto não deve ser tomado e a instrução a executar está no endereço PC + 4

- A instrução de *jump* difere do *beq* uma vez que é um salto absoluto e substitui os 28 bits menos significativos do PC. A sua implementação é diferente da do *beq*. (será analisada posteriormente)

Revisão do *datapath* (DP) do MIPS

- Junção dos vários blocos básicos do DP (**Tipo R + lw + sw**)

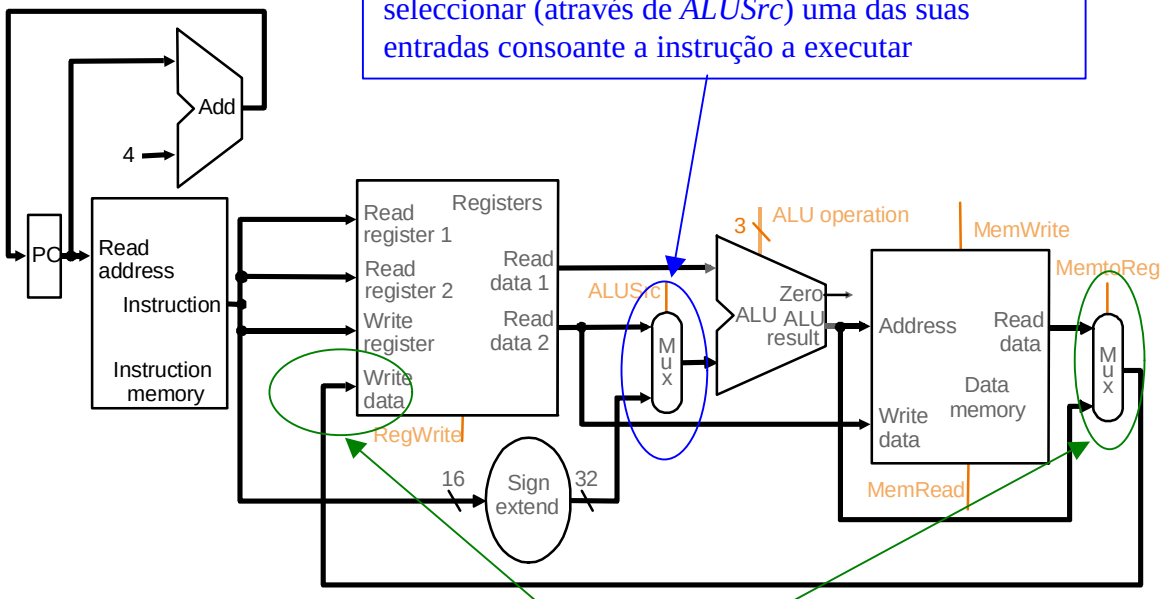


Select	C (saída)
0	A
1	B

A 32-bit wide 2-to-1 multiplexor

- Um **multiplexer** permite seleccionar uma saída entre várias entradas, através de um sinal de controlo (neste caso, designado por *select*)
- Uma mesma unidade do DP não pode ser usada várias vezes no mesmo ciclo.
- Em ciclos diferentes, as várias unidades podem ser utilizadas para instruções distintas (ex. a ALU e o banco de registos)

A segunda entrada da ALU pode provir do banco de registos (*Read data 2*, em instruções tipo R e *beq*) ou da unidade de extensão de sinal (instruções tipo I, nomeadamente *lw* e *sw*). O multiplexer permite seleccionar (através de *ALUSrc*) uma das suas entradas consoante a instrução a executar



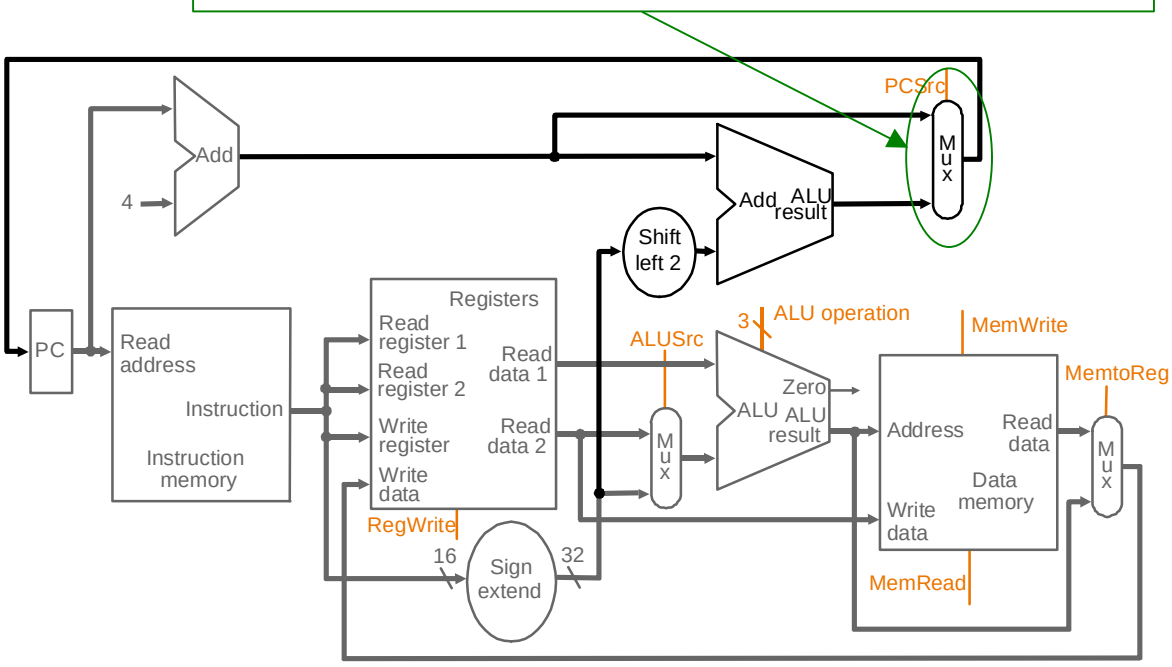
O valor que é escrito no banco de registos pode provir da memória (*lw*) ou da ALU (outras instruções)

 © João Luís Sobral 2002

Revisão do *datapath* (DP) do MIPS

- Junção dos vários blocos básicos do DP – **Suporte a *beq***

O suporte à instrução *beq* implica a introdução de mais um multiplexer, controlado pelo sinal *PCSrc*, para permitir a escolher o próximo valor do PC: execução sequencial (PC + 4) ou o valor calculador pelo *beq*.



- Sinais de controlo para a ALU

  - A tabela de verdade para a ALU:

ALU operation	Operação realizada
000	AND
001	OR
010	adição
110	subtracção
111	set on less than

  - A unidade *ALU Control* gera o sinal *ALUOperation* com base no sinal *ALUOp* e no campo *funct*, através da seguinte tabela:

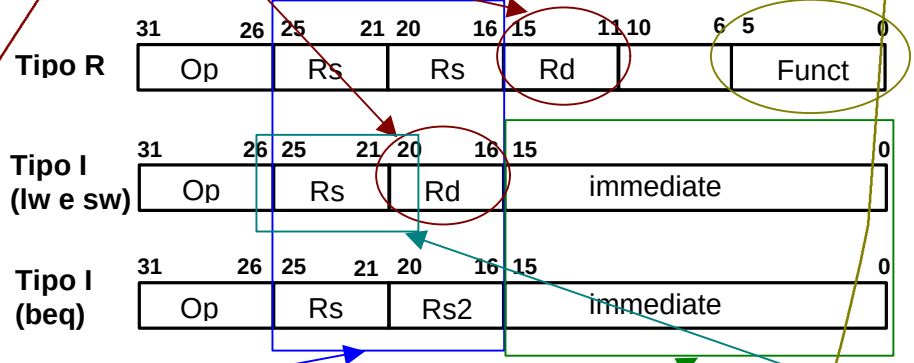
Instrução	*ALUOp*	*Campo funct*	Operação	*ALU operation*
LW	00	xxxxxx	adição	010
SW	00	xxxxxx	adição	010
BEQ	01	xxxxxx	subtracção	110
Tipo R (add)	10	100000	adição	010
Tipo R (sub)	10	100010	subtracção	110
Tipo R (and)	10	100100	and	000
Tipo R (or)	10	100101	or	001
Tipo R (slt)	10	101010	set on less than	111

Revisão do *datapath* (DP) do MIPS

- Finalização do DP – **Campos da Instrução**

O registo destino pode encontrar-se no campo [15-11] ou no campo [20-16], o que obriga à introdução de mais um multiplexer (*RegDst*)

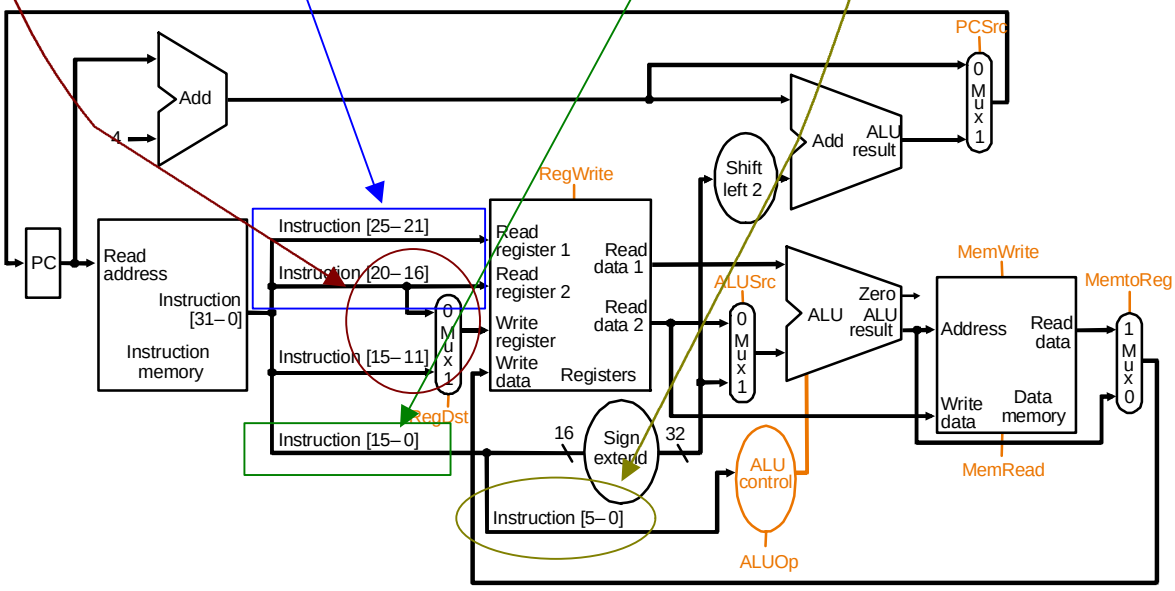
O campo da instrução [5-0] é utilizado pelo *ALU control* para gerar os sinais necessários para a ALU em instruções do tipo R.



O primeiro registo é sempre especificado no campo da instrução [25-21] enquanto o segundo registo está no campo [20-16], logo estes dois campos da instrução são ligados a *Read register 1* e *Read register 2*, respectivamente.

A constante (Imm16) de 16 bits está sempre no campo da instrução [15-0], logo este campo é ligado à unidade de extensão de sinal.

Quando é utilizada uma constante e um registo esse registo é especificado no campo [25-21]



Revisão do *datapath* (DP) do MIPS

- Suporte à instrução *jump* (*j end*)

Tipo J (J)

31 – 26	25 – 0
Op	address

A instrução *j* é um salto incondicional, cujo endereço destino é o endereço de 26 bits, incluído na instrução.

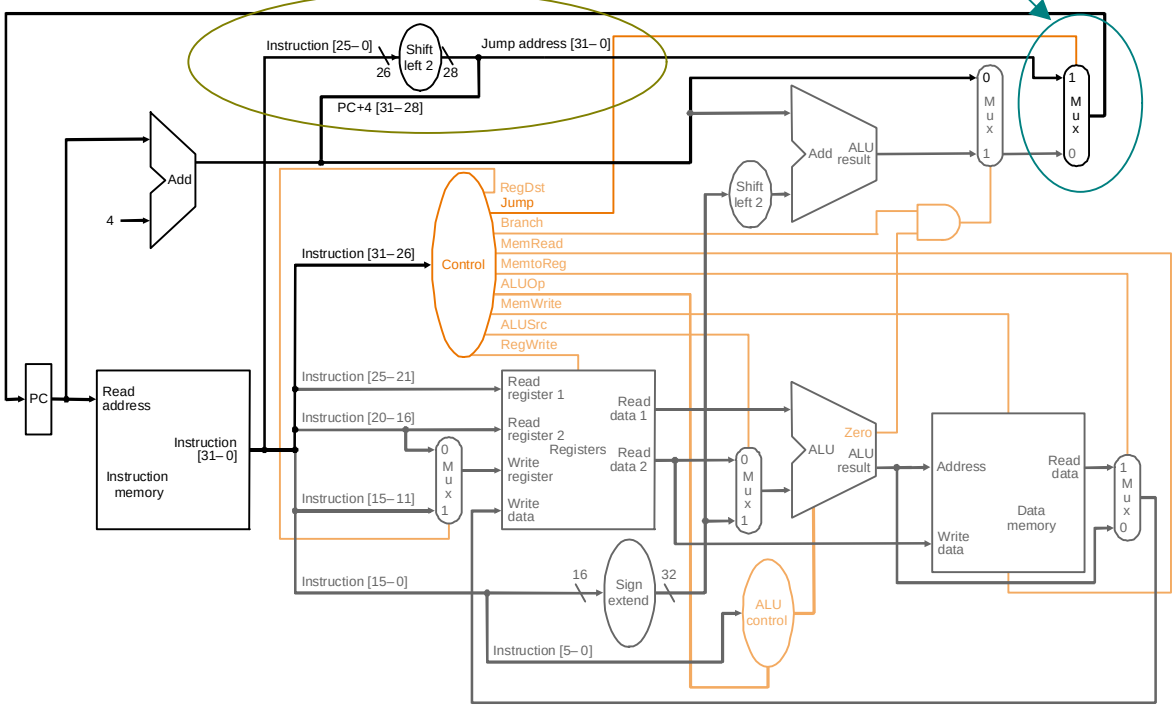
À constante de 26 bits são adicionados dois bits à direita (i.e., multiplicada por quatro).

Os restantes 4 bits são retirados de PC + 4

Endereço de salto

31 – 28	27 – 2	1 – 0
PC+4[31-28]	address	00

O suporte à instrução *j* requer a introdução de um multiplexer adicional e de um novo sinal de controlo (*jump*). Este multiplexer permite seleccionar o próximo valor do PC.



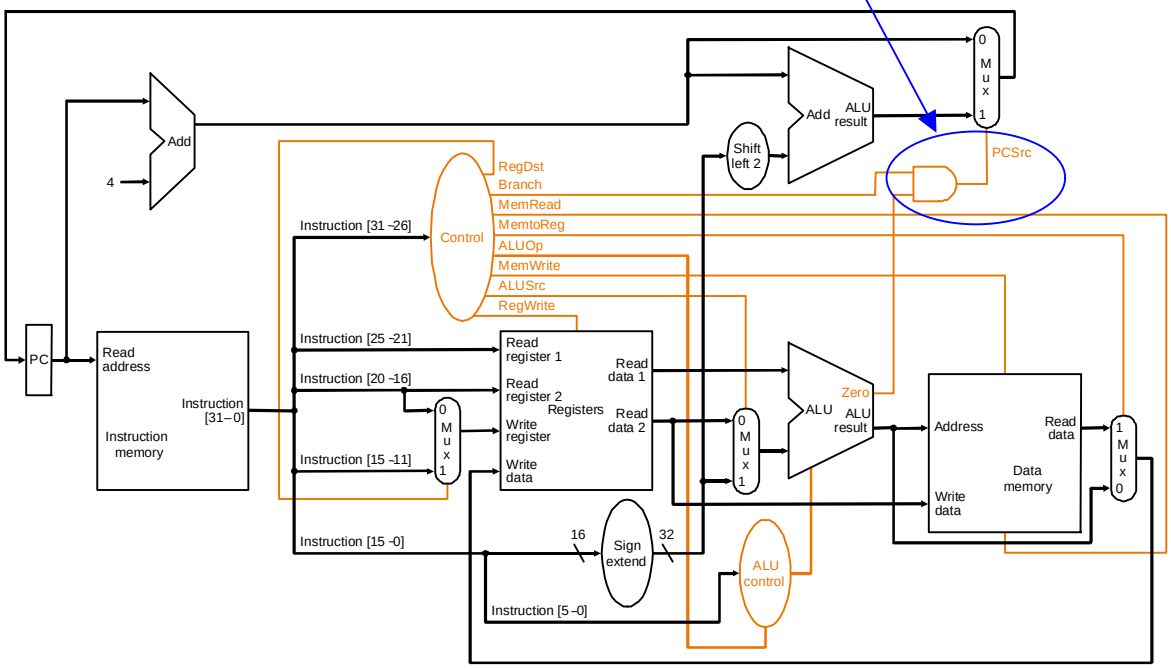
Revisão do *datapath* (DP) do MIPS

Unidade de controlo

- Resumo dos sinais de controlo:

Sinal	Efeito quando inactivo (0)	Efeito quando activo (1)
RegDst	O registo destino provém do campo rt (I[20-16])	O registo destino provém do campo rd (I[15-11])
RegWrite	nenhum	O valor em *Write data* é escrito no registo indicado em *Write register*
ALUSrc	O segunda entrada da ALU utiliza o valor do segundo registo (*Read data 2*)	O segunda entrada da ALU utiliza o campo Imm16, estendido para 32 bits
PCSrc	O próximo valor do PC é PC+4	O valor do PC é o destino do salto
MemRead	nenhum	O endereço de memória é lido e colocado em *Read data*
MemWrite	nenhum	Os dados em *Write data* são escritos no endereço de memória
MemtoReg	Os dados a escrever num registo provêm da ALU	Os dados a escrever num registo provêm da memória

O sinal *PCSrc* deve ser activado quando a saída *Zero* da ALU está activa e a instrução em execução é um *beq*, o que é implementado através do AND de um novo sinal *branch* (calculado do *opcode* da instrução) com *Zero*.



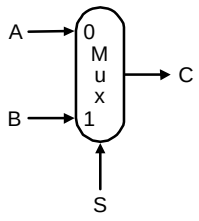
- A unidade de controlo pode ser resumida na seguinte tabela:

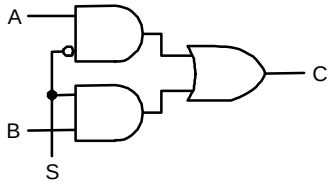
Instrução	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU op
Tipo R	1	0	0	1	0	0	0	10
lw	0	1	1	1	1	0	0	00
sw	X	1	X	0	0	1	0	00
beq	X	0	X	0	0	0	1	01

Implementação do DP do MIPS

- *Multiplexer*

A	B	S	C
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1





- $C = (A.\overline{S}) + (B.S)$

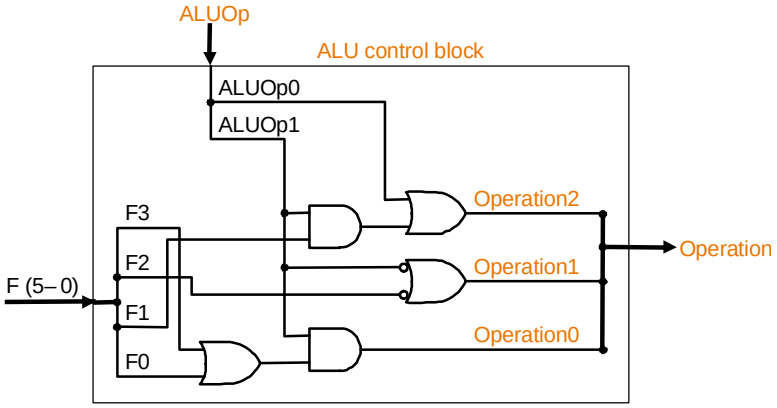
- *ALU Control*

Instrução	*ALUOp*	Campo funct	Operação	*ALU operation*
LW, SW	00	xxxxxx	adição	010
BEQ	01	xxxxxx	subtracção	110
Tipo R (add)	10	100000	adição	010
Tipo R (sub)	10	100010	subtracção	110
Tipo R (and)	10	100100	and	000
Tipo R (or)	10	100101	or	001
Tipo R (slt)	10	101010	set on less than	111

- Trata-se de uma função com 8 entradas (2 bits de *ALUOp* e 6 bits de *Funct*) e 3 saídas:

ALUOp		Funct						*ALU operation*
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

- Após algumas simplificações (nomeadamente, intruduzindo *don´t cares* adicionais) a função pode ser implementada pelo seguinte circuito:



Implementação do DP do MIPS

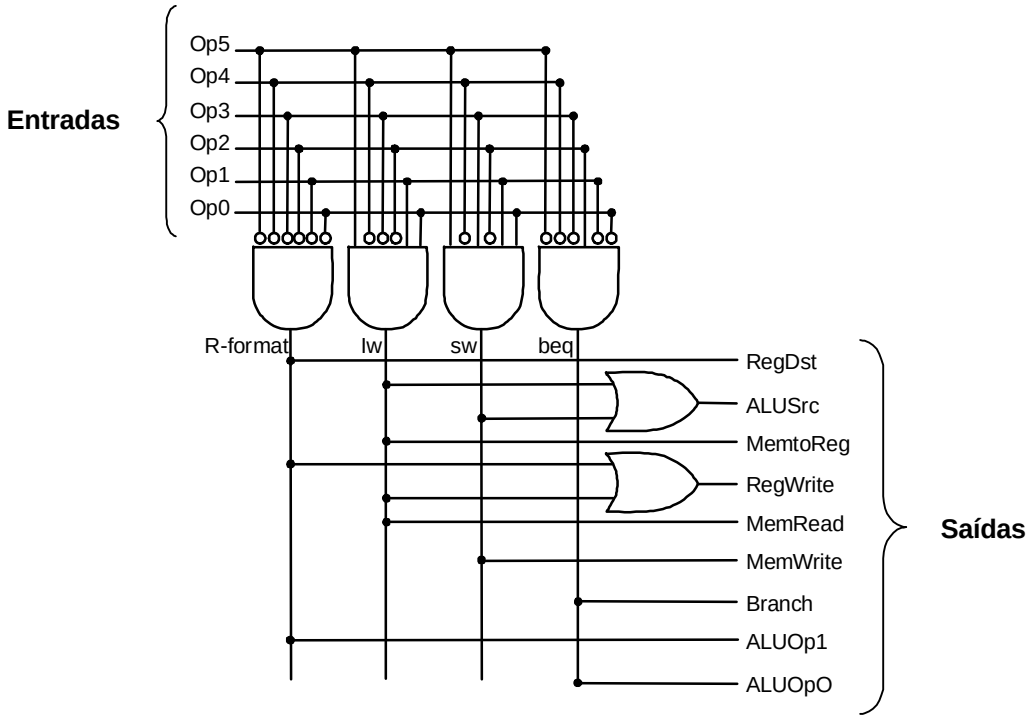
- Unidade de controlo

Instrução	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU op
Tipo R	1	0	0	1	0	0	0	10
lw	0	1	1	1	1	0	0	00
sw	X	1	X	0	0	1	0	00
beq	X	0	X	0	0	0	1	01

- Trata-se de uma função com 6 entradas (6 bits de *opcode* da instrução) e 9 saídas (8 sinais de controlo):

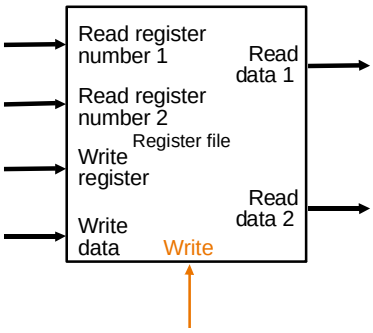
OpCode Op[5-0]	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU op
000000	1	0	0	1	0	0	0	10
100011	0	1	1	1	1	0	0	00
101011	X	1	X	0	0	1	0	00
000100	X	0	X	0	0	0	1	01

- A implementação com uma PLA (um *array* ANDs seguido por um *array* de ORs):

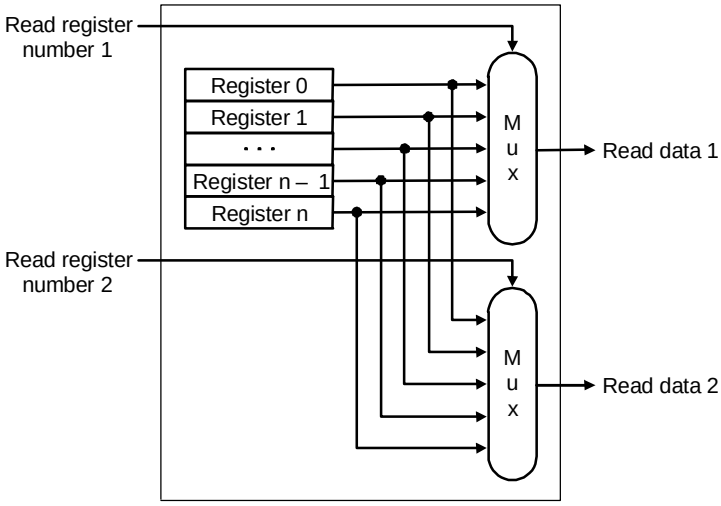


Implementação do DP do MIPS

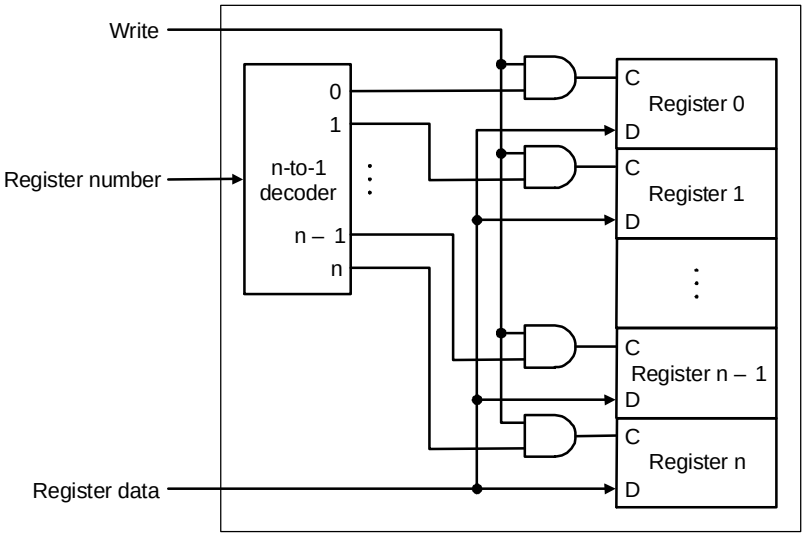
- Banco de registos



- Os registos podem ser implementados através de flip-flops do tipo D.
- A leitura é implementada por um multiplexer que selecciona o registo cujo valor é apresentado na saída:



- A escrita é implementada através de um descodificador:

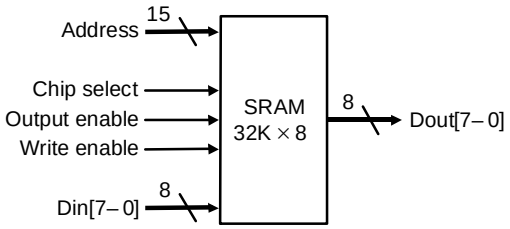


Implementação do DP do MIPS

Memória SRAM (*static Random access memory*)

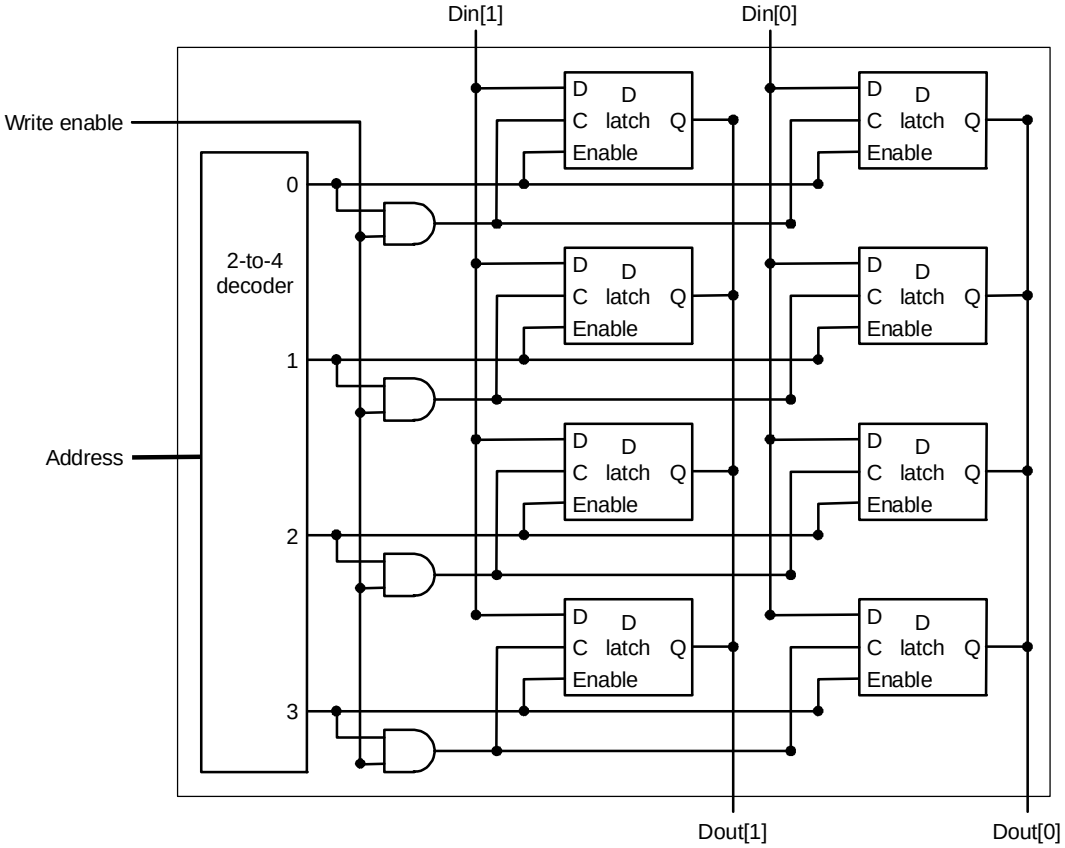
- A SRAM possui uma configuração específica, geralmente caracterizada pelo número de localizações endereçáveis e a largura de cada endereço (i.e., número de bits de cada célula).

- Por exemplo, uma SRAM de 32K x 8 possui 32K células, cada uma com 8 bits. Neste caso são necessárias 15 linhas de endereço:



- A SRAM pode ser implementada por *D latches*, com uma entrada *Enable* para controlar a saída *three-state*.

- Exemplo de uma 4 x 2 SRAM:

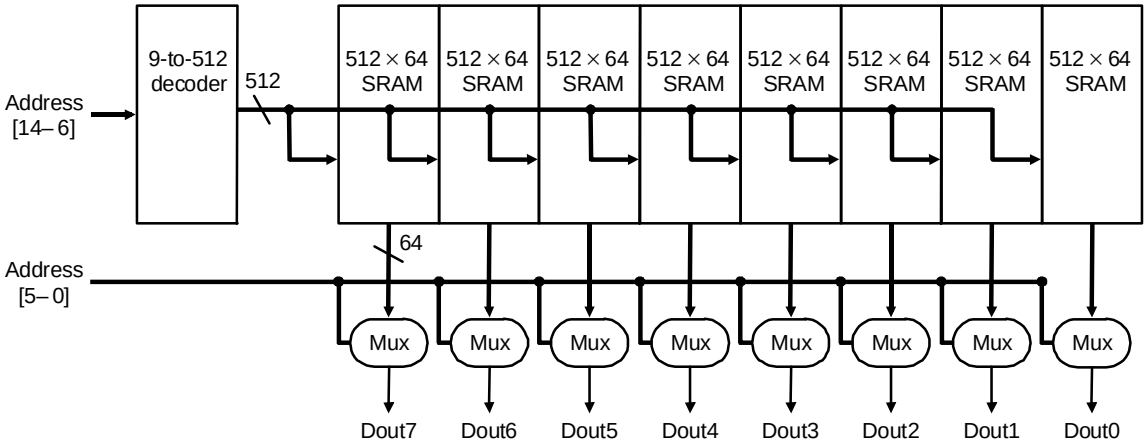


Implementação do DP do MIPS

Memória SRAM (*static random access memory*)

- Os descodificadores utilizados na SRAM podem ter uma dimensão elevada. Por exemplo uma SRAM de 16K x 8 necessita de um descodificador 14 para 16K!

- Para diminuir a complexidade do descodificadores, pode ser utilizada uma descodificação a dois níveis:



- Por exemplo, esta configuração evita a utilização de um descodificador 15 para 32K ou de um multiplexer de 32K para 1.

- A SRAM é caracterizada pelo tempo de acesso, que varia entre 5 ns e 25 ns (em 1997).

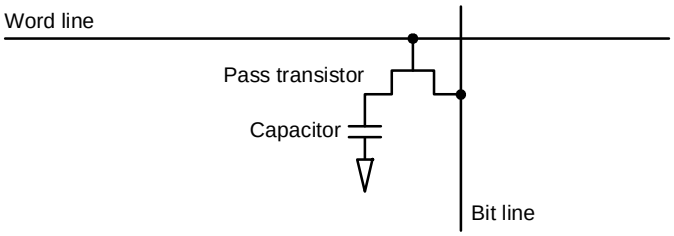
- Recentemente foi introduzida SRAM síncrona (e também DRAM) que permitem transferências em *burst* de endereços sequenciais, o que aumenta o débito da memória.

Implementação do DP do MIPS

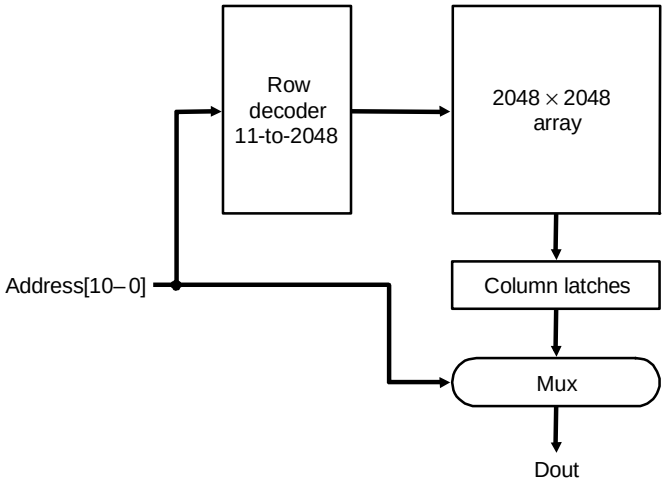
Memória DRAM (*dynamic random access memory*)

- Ao contrário da SRAM, a DRAM apenas mantém um valor numa célula durante um período de tempo, sendo necessário "refrescar" esse valor com intervalos de milisegundos.

- Uma célula de DRAM é constituída por um transistor e um condensador, o que permite atingir densidades por chip muito superiores à da SRAM (que requer entre 4 e 6 por célula).



- A DRAM utiliza uma descodificação em dois níveis, constituída por um acesso a uma linha seguido pela selecção da coluna dessa linha.



- Os sinais de linha e coluna utilizam os mesmos pinos do chip de memória. O par de sinais RAS (Row Access Stobe) e CAS (Column Access Strobe) é utilizado para destinguir os dois tipos de sinais.

- As memórias DRAM são bastante mais lentas que as SRAM (um factor entre 5 e 10). Os tempos de acesso variam entre 60ns e 110ns (em 1997)

- Nas memórias recentes, devido à utilização de *burst* os acesso a elementos da mesma linha são substancialmente inferiores (cerca de 25ns)

- As memórias DRAM-ECC contêm bits adicionais redundantes que permitem detectar e corrigir erros.

Bibliografia

- **Secções 5.1, 5.2, 5.3, C2** e **B5** de *Computer Organization and Design: the hardware /software interface*, D. Patterson and J. Henessy, Morgan Kaufmann, 2ª edição, 1998.

- **Adicional:**

  - Quem não estiver suficientemente familiarizado com o conjunto de instruções e com o código máquina do MIPS deve ler previamente todo o capítulo 3 do livro.