Arquitetura de Computadores

Estrutura interna de um processador de 8 bits – Arquitetura de Harvard

Estrutura interna de RAM estática e ROM

Bib: C – secções 4.1 e 4.3

João Pedro Patriarca (jpatri@cc.isel.ipl.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Caso de estudo

Baseado no exercício realizado na sessão anterior

- CPU com uma arquitetura de 8 bits do tipo RISC
- Banco de registos constituído por 4 registos genéricos de 8 bits cada
- ALU de 8 bits constituída pelas operações adição e subtração

Seletor	Operação
0	A+B
1	A-B

- Registo CPSR com a *flag* Z
- Capacidade de endereçamento de 256 bytes com acesso apenas ao byte

Caso de estudo
Conjunto de instruções

Instrução	Descrição
1. add rd, rn	rd ← rd + rn
2. b rn	PC ← rn
3. bzs label	(CPSR.Z == 1) ? PC ← PC + label : PC ← PC + 1
4. cmp rn, rm	(rn – rm == 0) ? CPSR.Z ← 1 : CPSR.Z ← 0
5. ldr rd, [rn]	rd ← mem[rn]
6. mov rd, #imm3	rd ← imm3
7. str rs, [#imm3]	mem[rn] ← rs

- A definição do formato das instruções (ISA) e respetivas codificações foram realizadas no âmbito do exercício anterior

Formato e codificação do conjunto de instruções

	7	6	5	4	3	2	1	0
1. add rd, rn	0	0	0	-	rn		rd	
2. b rn	0	1	0	-	rn		-	-
3. bzs label	0	1	1	label				
4. cmp rn, rm	0	0	1	-	rn		rm	
5. ldr rd, [rn]	1	0	0	-	rn		rd	
6. mov rd, #imm3	1	1	1	imm3			rd	
7. str rs, [#imm3]	1	0	1	imm3			rs	

opcode = bits(7..5)

imm3 = bits(4..2)

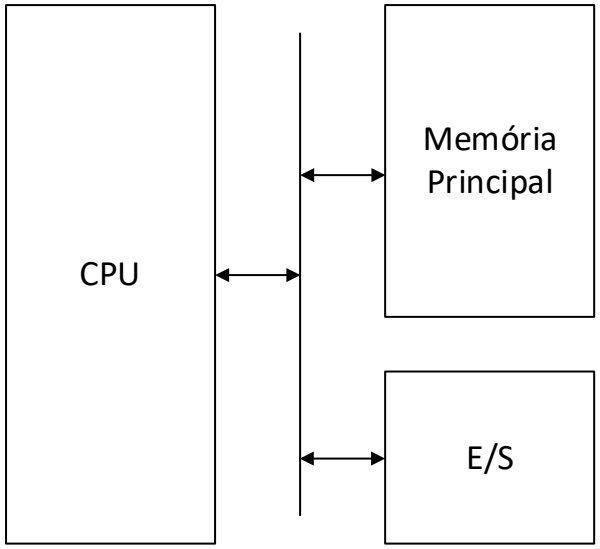
label = bits(4..0)

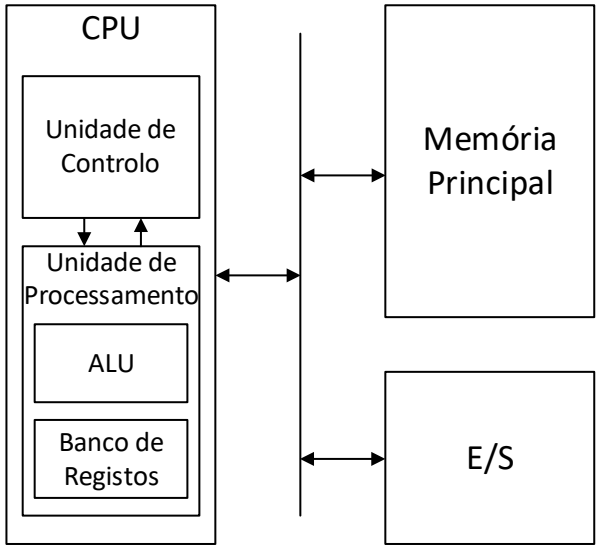
DA (*Destination Address*) = BA (*operand B Address*) = bits(1..0)

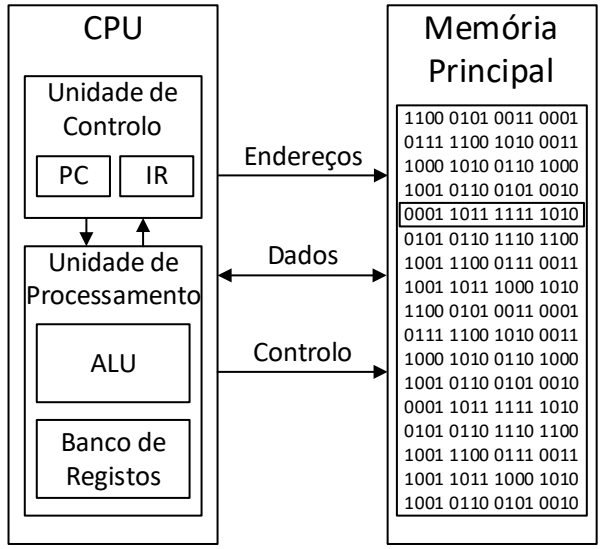
AA (*operand A Address*) = bits(3..2)

OPALU (*ALU operation*) = bit(5)

Sistema computacional genérico

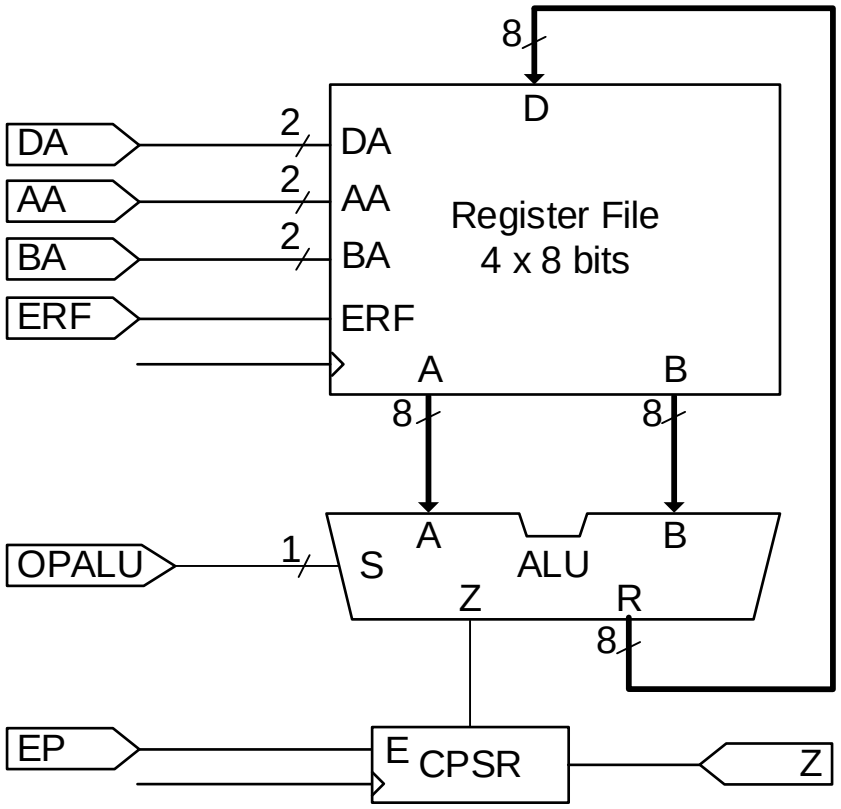






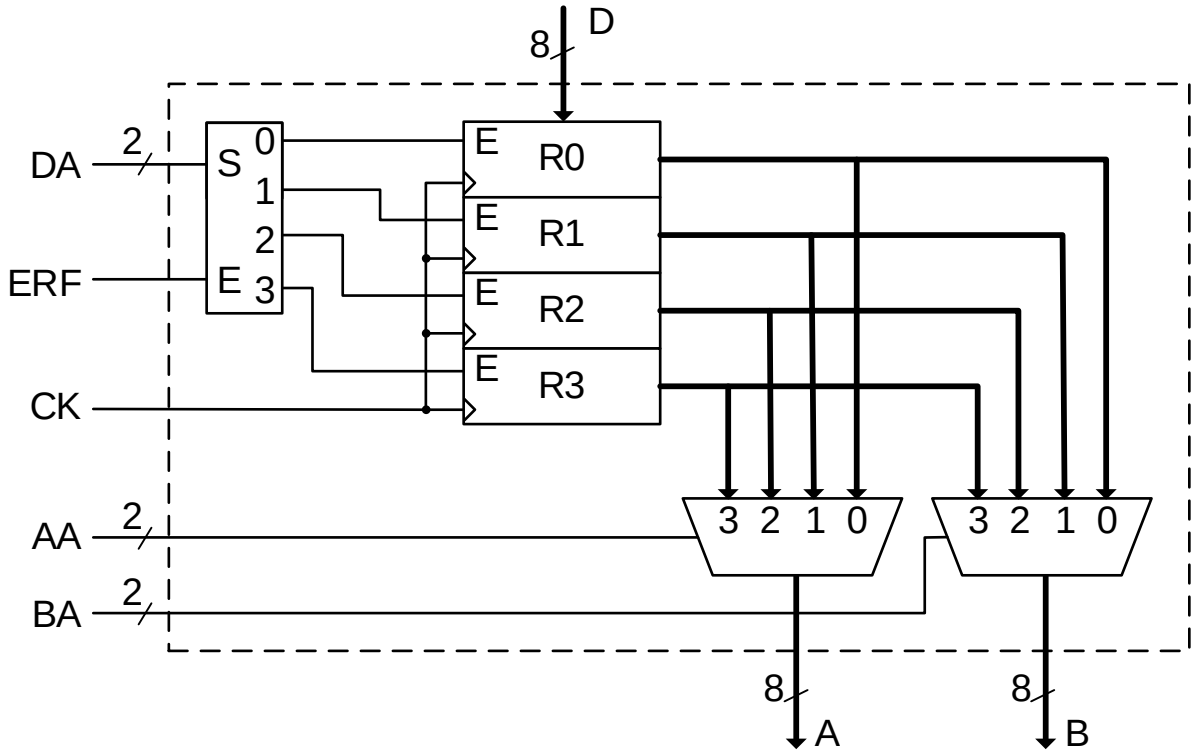
Unidade de processamento

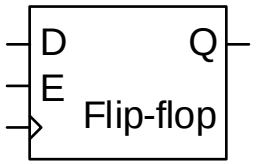
Suporte para as instruções `add rd, rn` e `cmp rn, rm`

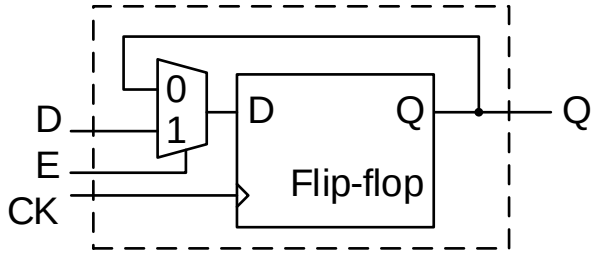


- Legenda do banco de registos:
  - D (*Data*): 8 bits de entrada com o valor a escrever em um dos quatro registos
  - A (*A output*): 8 bits de saída com o valor de um dos quatro registos
  - B (*B output*): 8 bits de saída com o valor de um dos quatro registos
  - DA (*Destination Address*): registo que poderá ser escrito
  - AA (*A Address*): registo cujo valor é colocado na saída A do RegFile
  - BA (*B Address*): registo cujo valor é colocado na saída B do RegFile
  - ERF (*Enable Register File*): habilita ou inibe a escrita em um dos quatro registos
- Perguntas sobre o banco de registos:
  - São precisas duas saídas (A e B)?
  - Os registos do banco de registos precisam ser síncronos?
  - São precisos três barramentos de endereçamento (DA, AA e BA)?

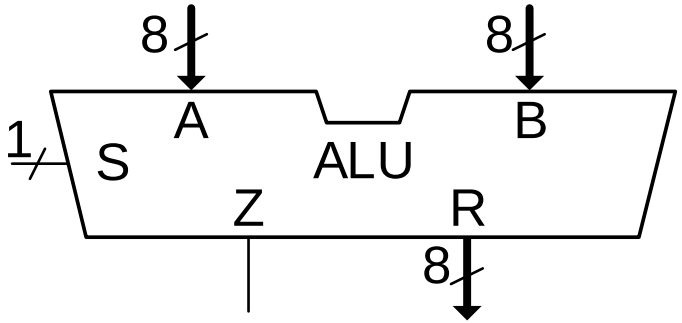
Banco de registos
Estrutura interna

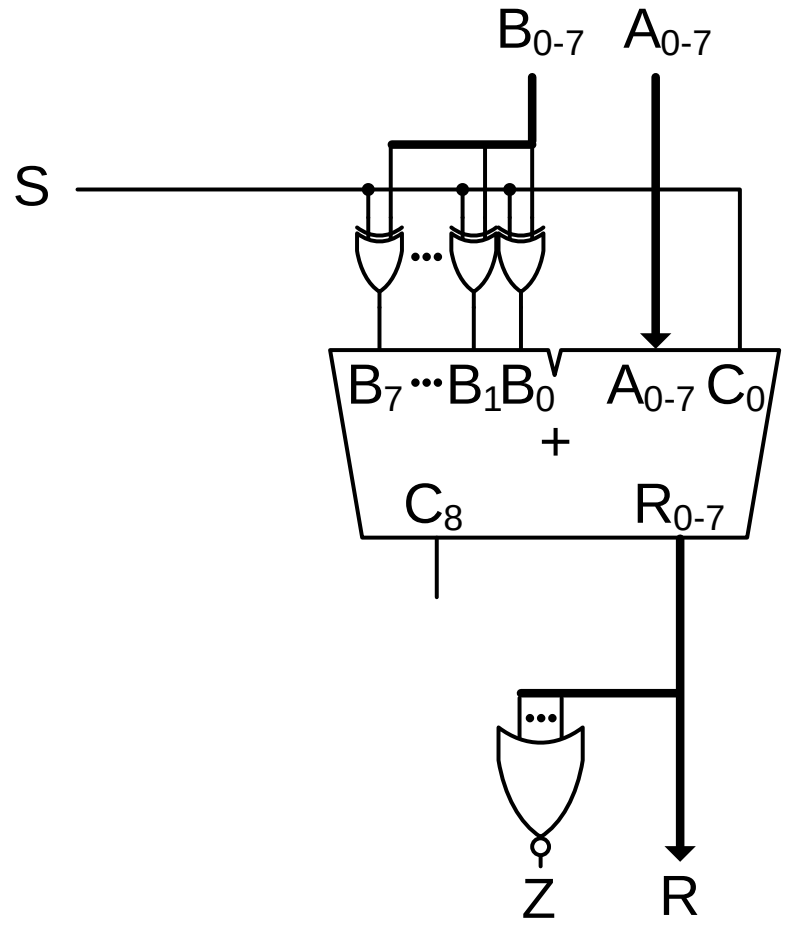






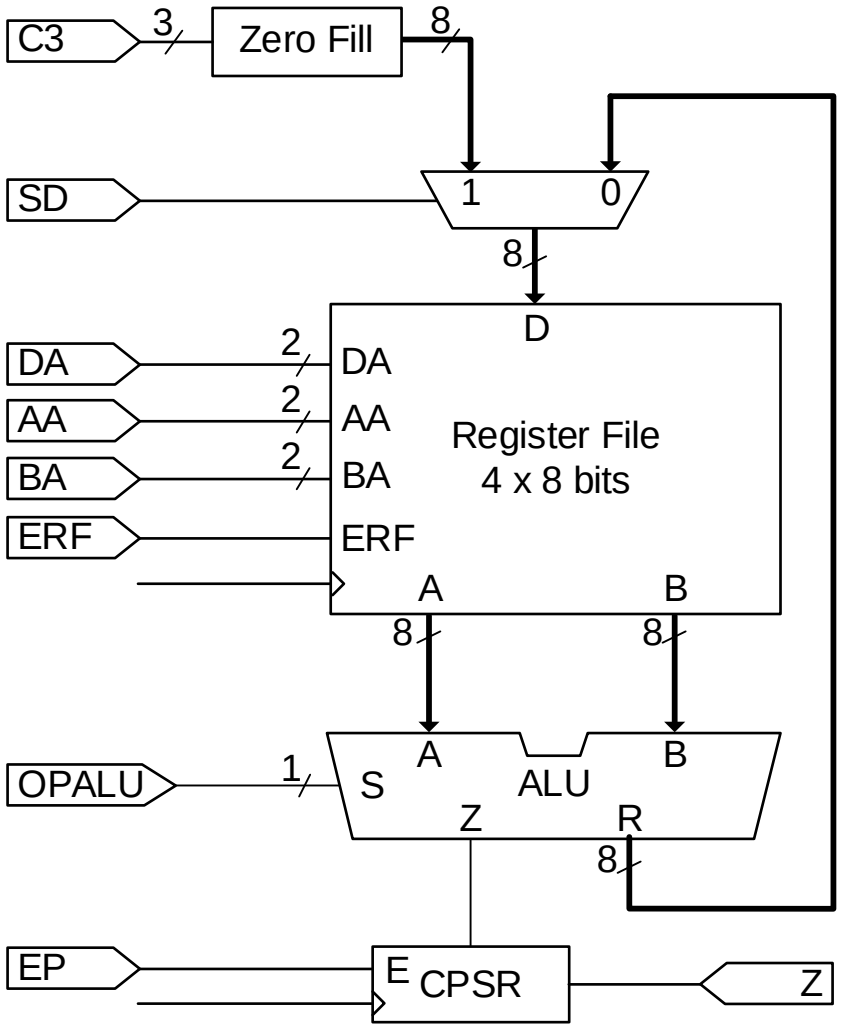
Unidade aritmética
Estrutura interna

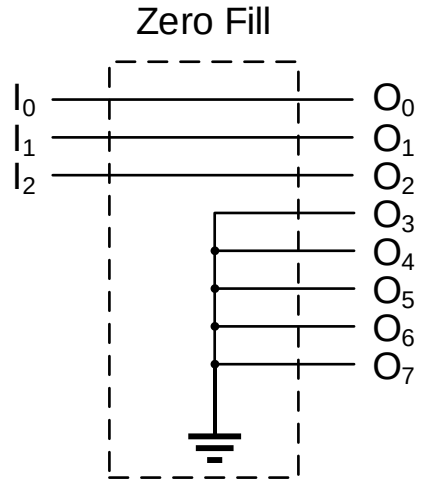




Unidade de processamento

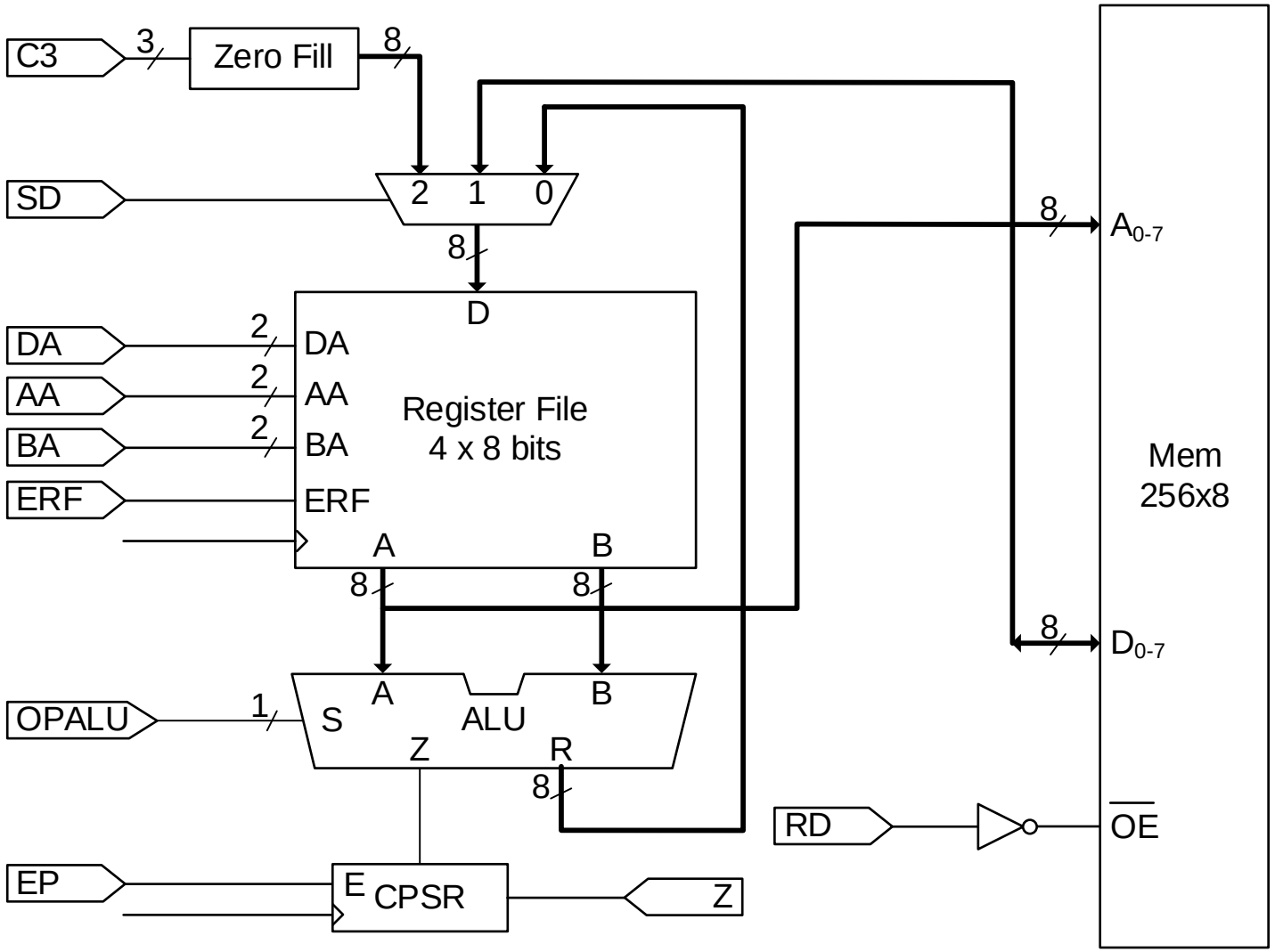
Suporte para a instrução `mov rd, #imm3`





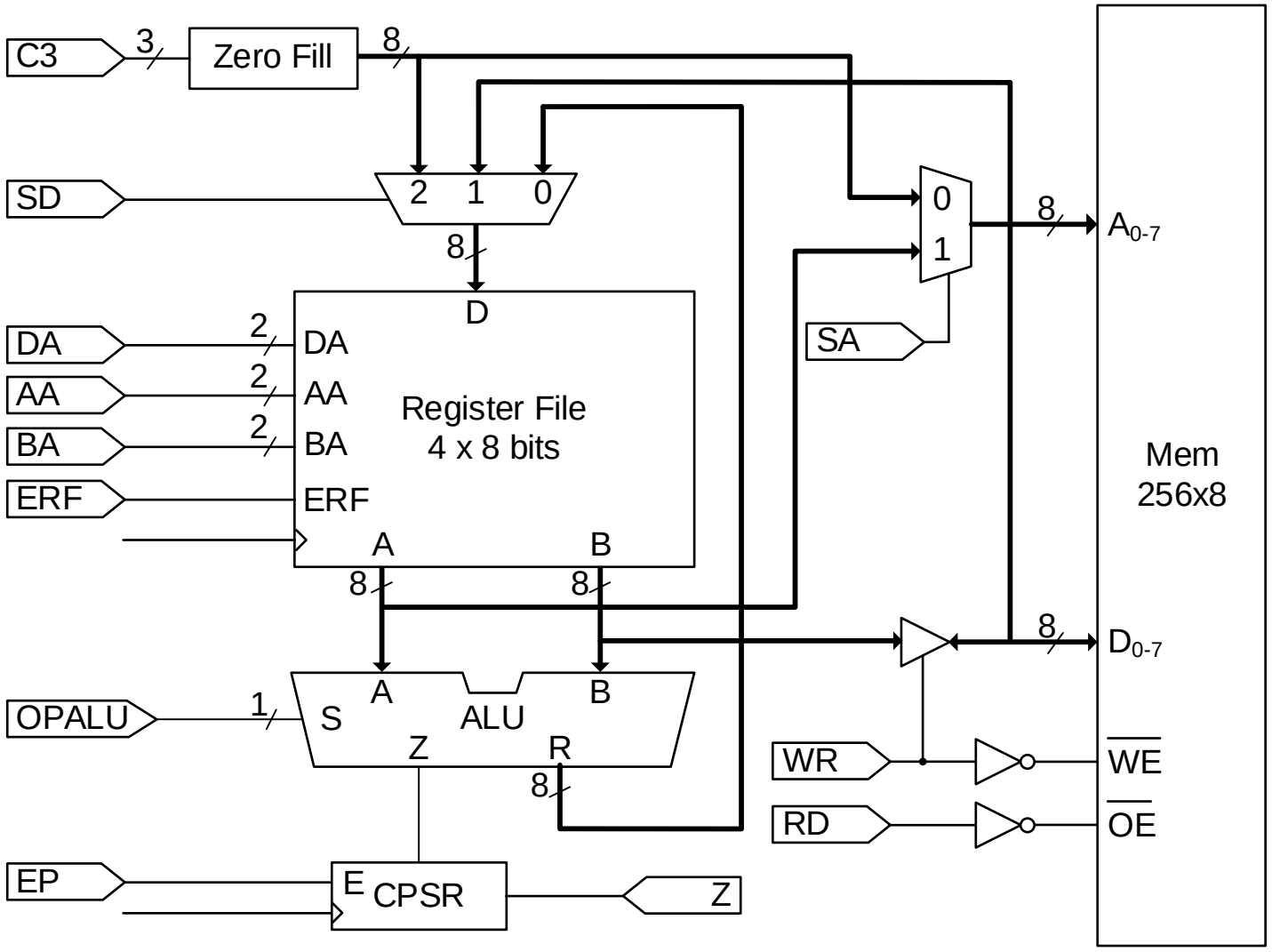
Unidade de processamento

Suporte para a instrução `ldr rd, [rn]`

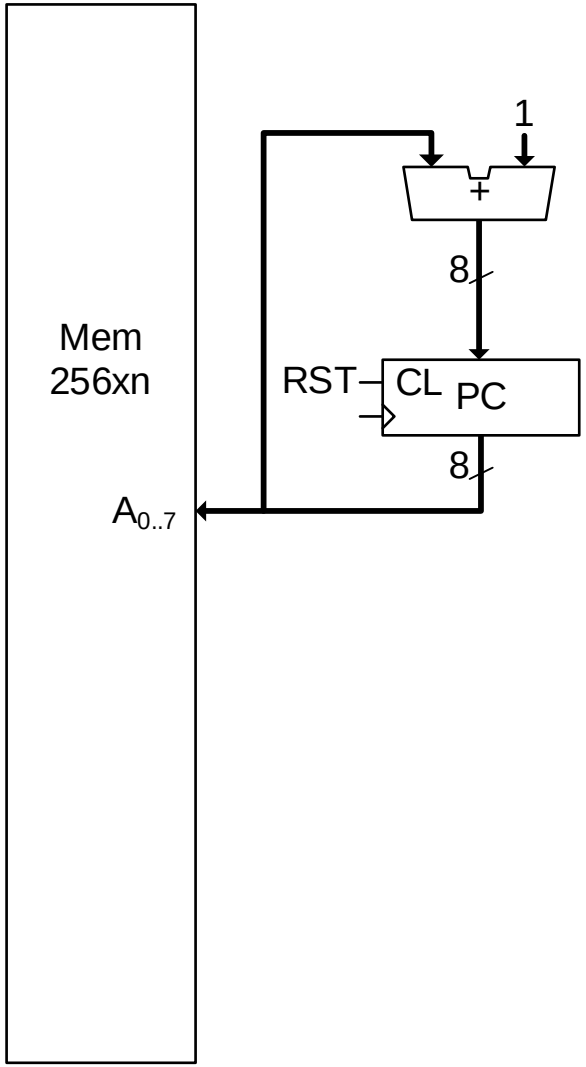


Unidade de processamento

Suporte para a instrução `str rd, [#imm3]`

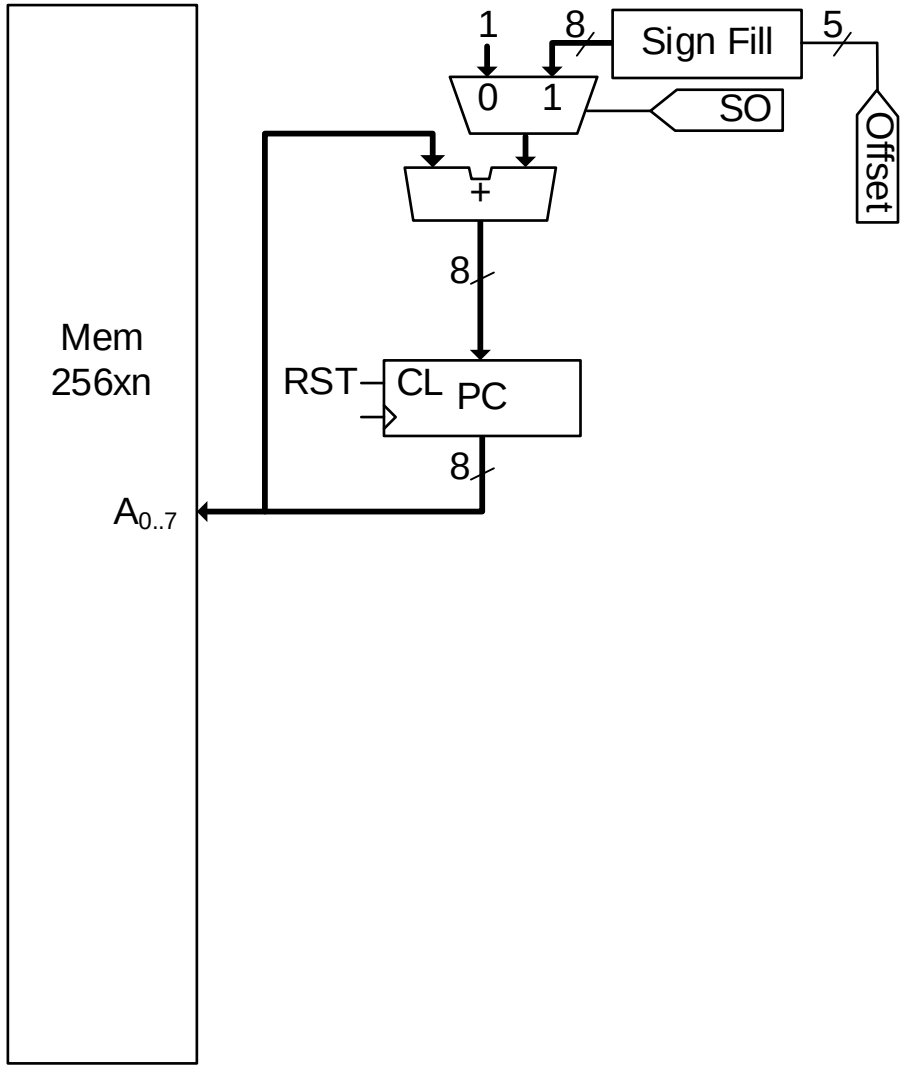


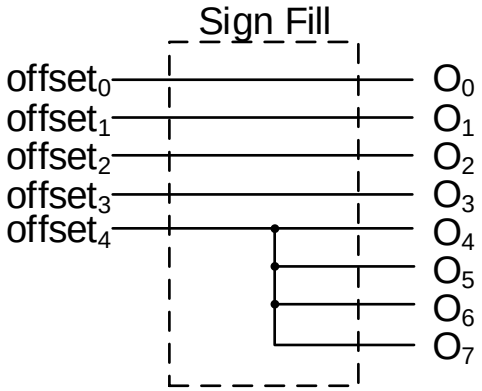
Unidade de controlo
Sequenciador de programa



Unidade de controlo

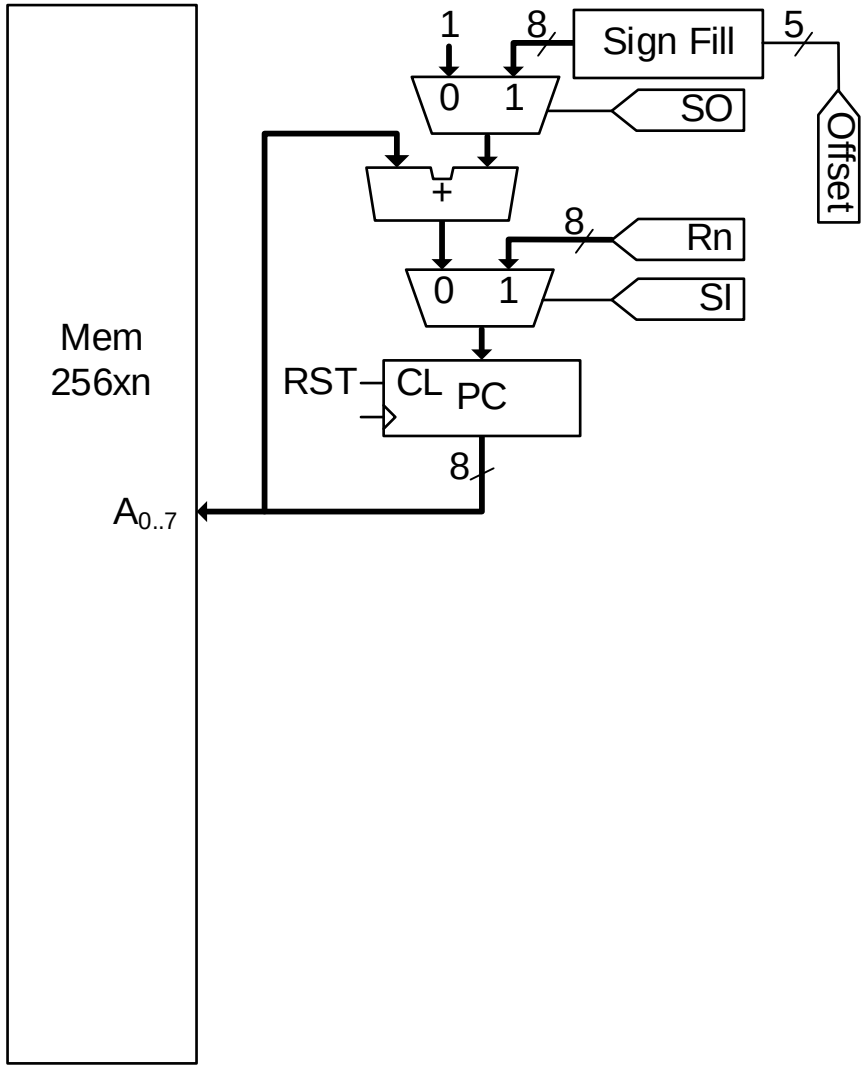
Suporte para a instrução `bzs label`



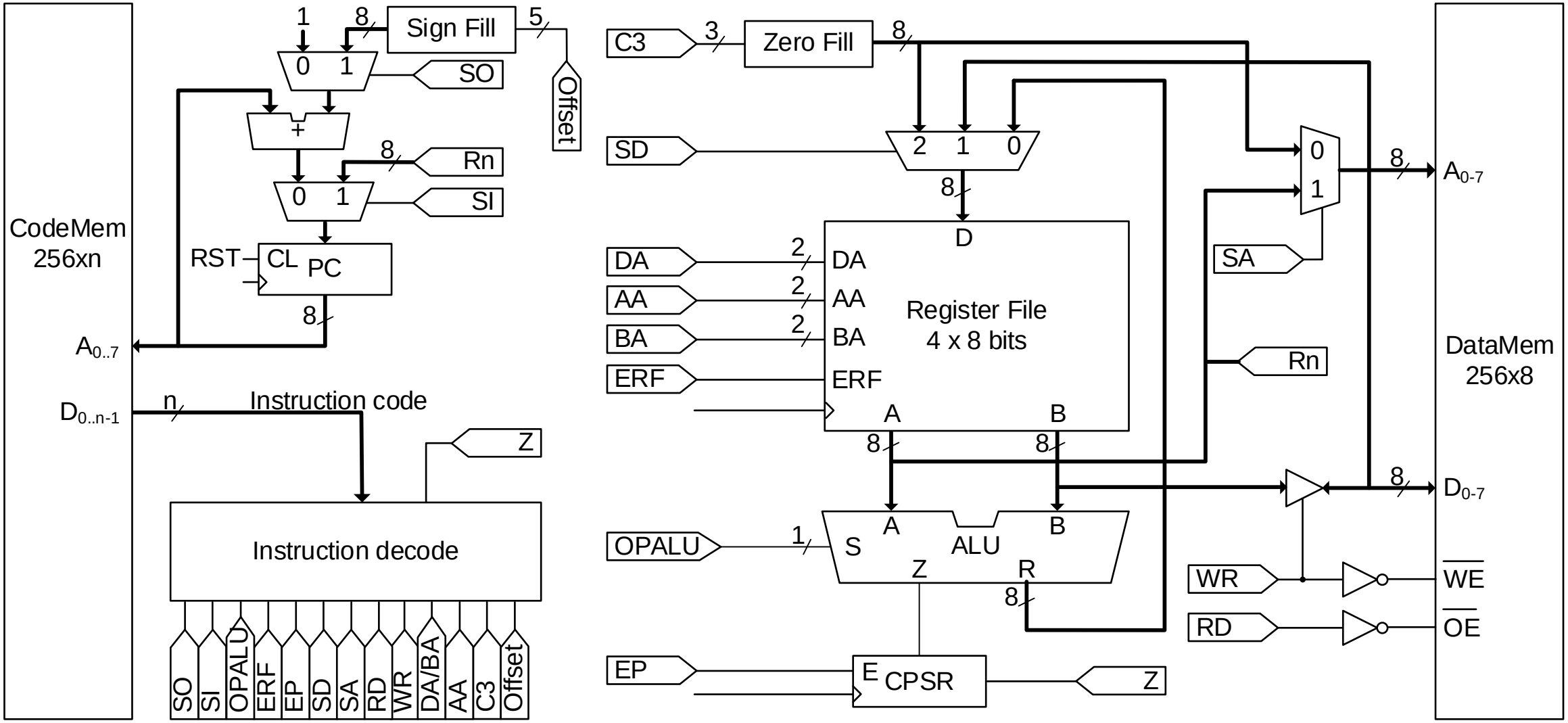


Unidade de controlo

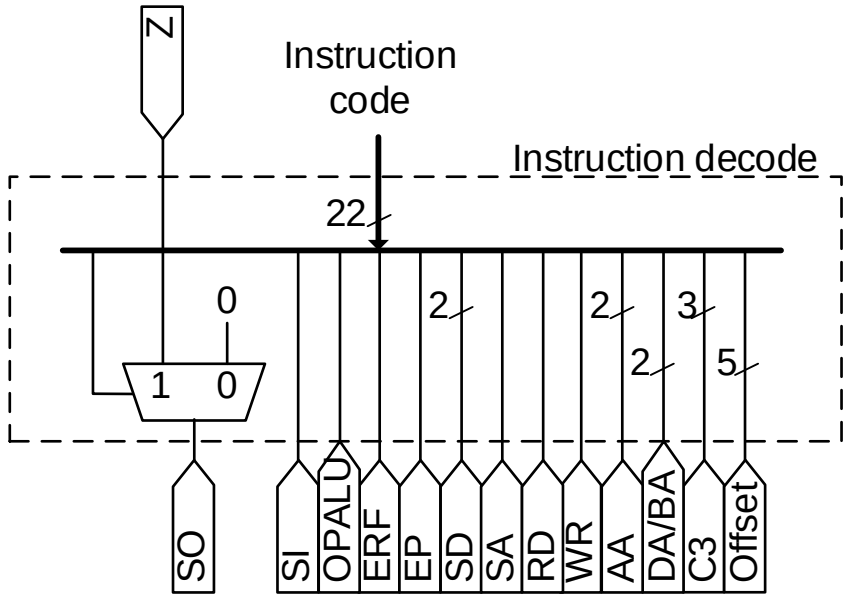
Suporte para a instrução `b rn`



Unidade de controlo
Descodificador de instruções



Descodificador de instruções
Sem compressão do código das instruções

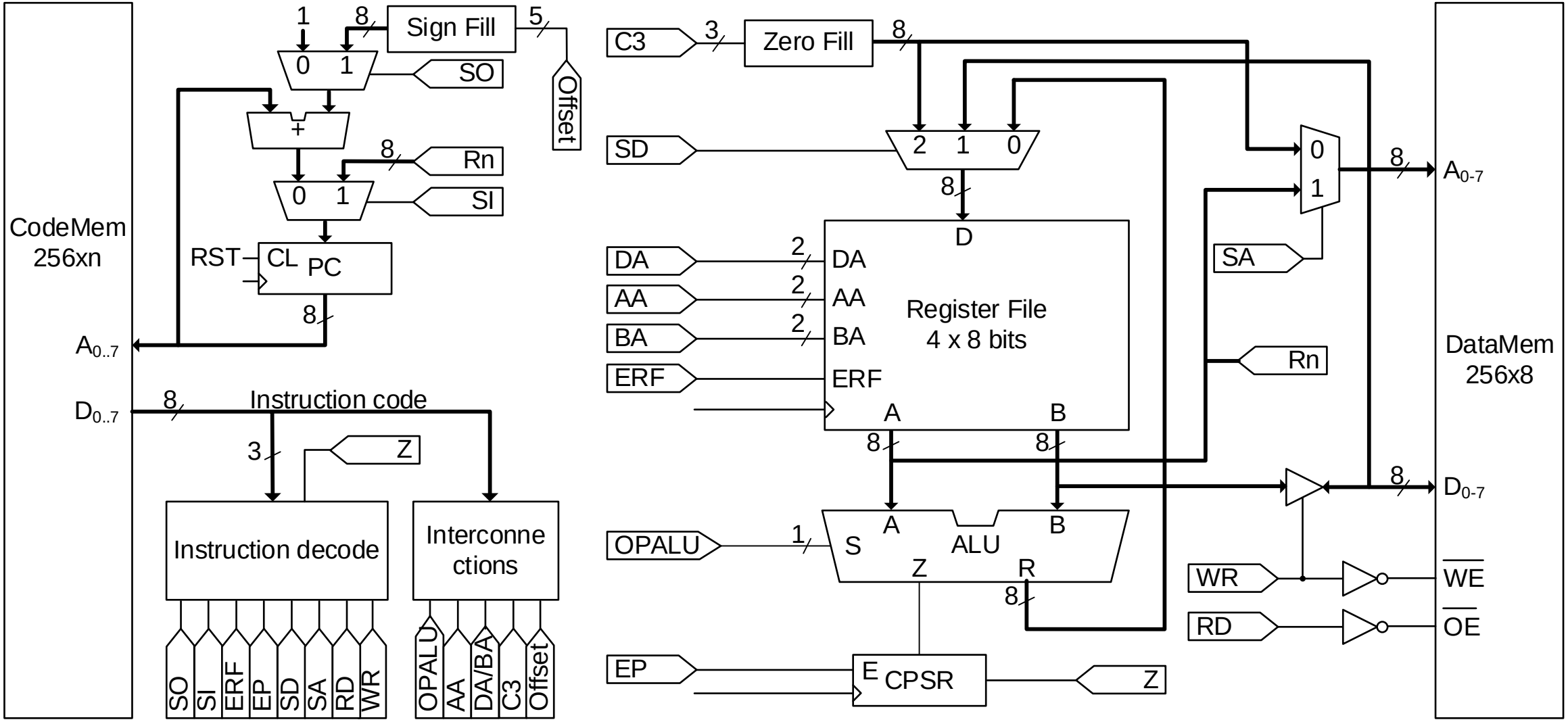


- Exemplos de codificações de instruções:

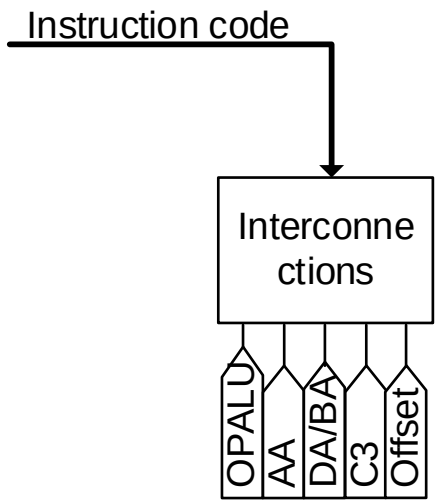
	21					16							8									0
	_SO	SI	AU	RF	EP	SD		SA	RD	WR	AA		DA/BA		C3			Offset				
add r2, r3	0	0	0	1	1	0	0	-	0	0	1	1	1	0	-	-	-	-	-	-	-	-
cmp r1, r2	0	0	1	0	1	-	-	-	0	0	0	1	1	0	-	-	-	-	-	-	-	-
mov r0, #5	0	0	-	1	0	1	0	-	0	0	-	-	0	0	1	0	1	-	-	-	-	-
ldr r2, [r0]	0	0	-	1	0	0	1	1	1	0	0	0	1	0	-	-	-	-	-	-	-	-
str r1, [#4]	0	0	-	0	0	-	-	0	0	1	-	-	0	1	1	0	0	-	-	-	-	-
b r3	-	1	-	0	0	-	-	-	0	0	1	1	-	-	-	-	-	-	-	-	-	-
bzs -2	1	0	-	0	0	-	-	-	0	0	-	-	-	-	-	-	-	1	1	1	1	0

+ Controlo dos módulos funcionais diretamente a partir do código das instruções
− Inadequado o número de bits para codificar instruções: 22 bits

Descodificador de instruções
Com compressão do código das instruções

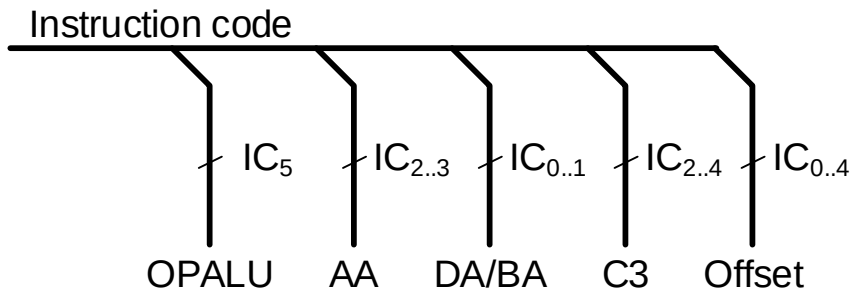


Bloco *Interconnections*

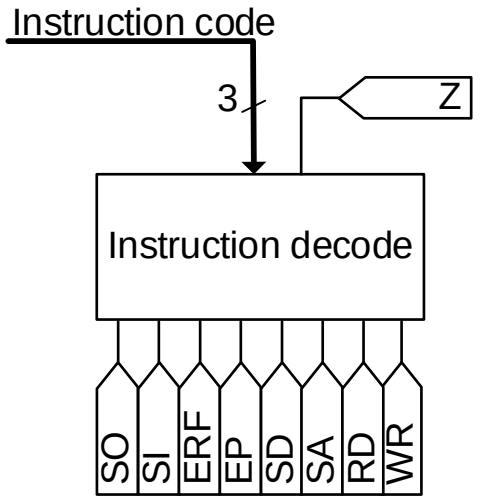


	7	6	5	4	3	2	1	0
1. add rd, rn	0	0	0	-	rn		rd	
2. b rn	0	1	0	-	rn		-	-
3. bzs label	0	1	1	label				
4. cmp rn, rm	0	0	1	-	rn		rm	
5. ldr rd, [rn]	1	0	0	-	rn		rd	
6. mov rd, #imm3	1	1	1	imm3			rd	
7. str rs, [#imm3]	1	0	1	imm3			rs	

			ALU		AA		DA/BA	
OPCODE				C3				
				Offset				



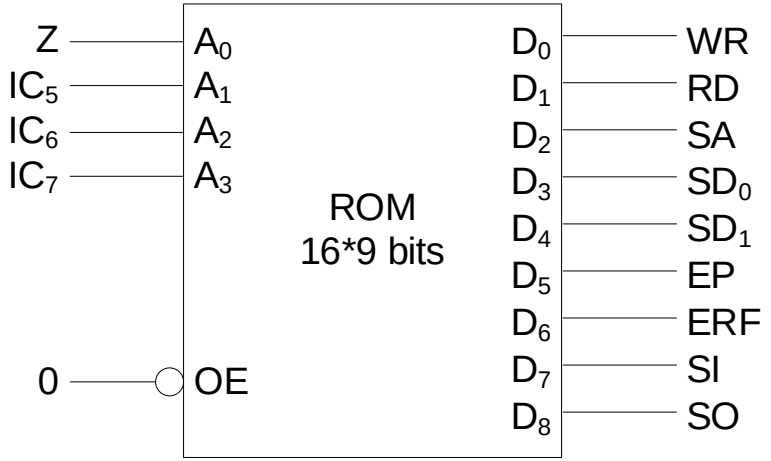
Bloco *Instruction decode*



- Circuitos combinatórios: as saídas do descodificador de instruções dependem exclusivamente do código de operação com exceção da saída SO que depende do código de operação e do valor da *flag* Z
- Exemplos para algumas saídas

$$EP = \overline{IC_7}.\overline{IC_6}.\overline{IC_5} + \overline{IC_7}.\overline{IC_6}.IC_5$$
$$SA = IC_7.\overline{IC_6}.\overline{IC_5}$$
$$SO = \overline{IC_7}.IC_6.IC_5.Z$$

- Implementação com uma ROM



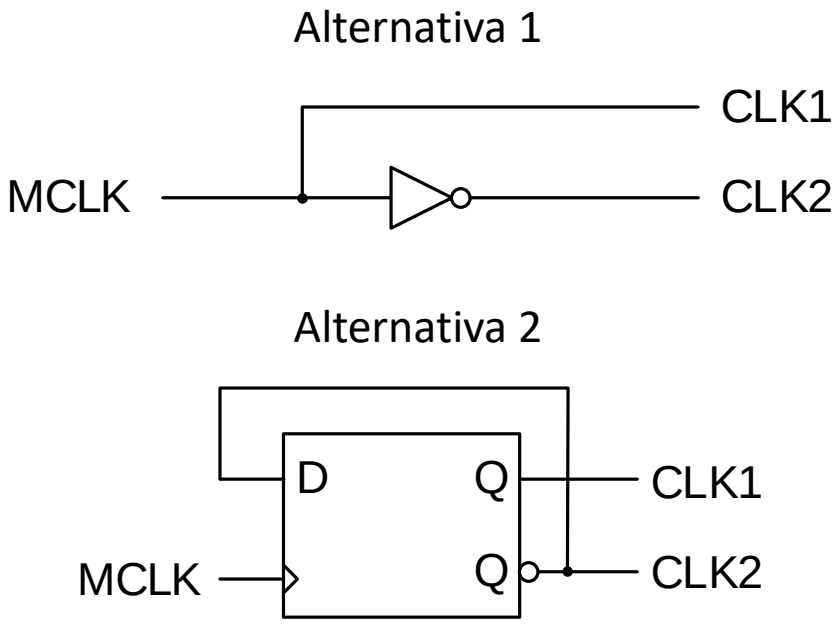
Programação da ROM: micro-código

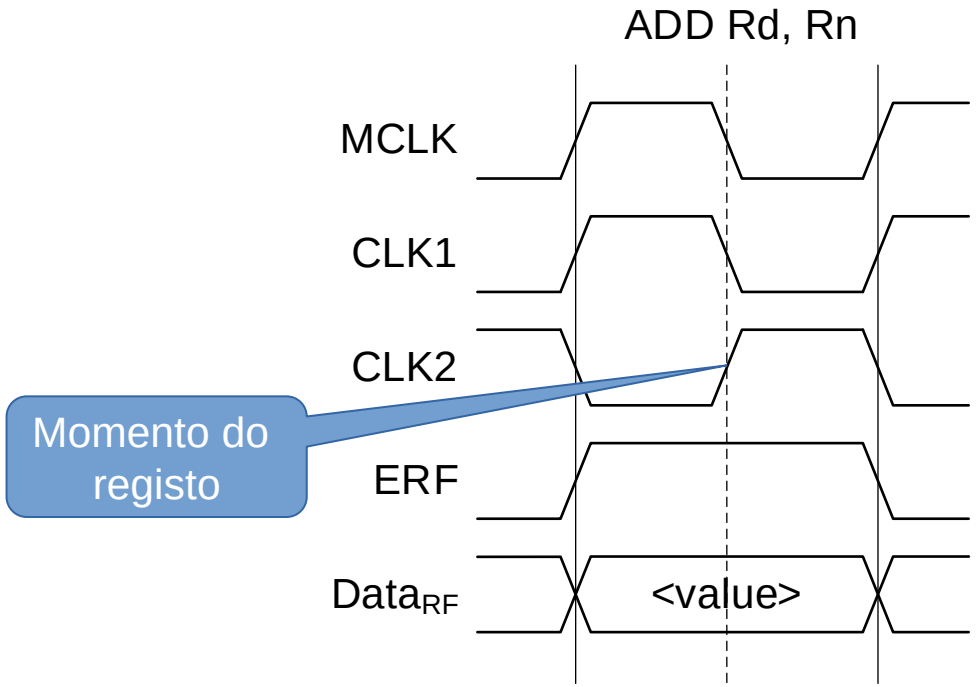
	IC7	IC6	IC5	Z	SO	SI	ERF	EP	SD1	SD0	SA	RD	WR
	A3	A2	A1	A0	D8	D7	D6	D5	D4	D3	D2	D1	D0
`add rd, rn`	0	0	0	-	0	0	1	1	0	0	-	0	0
`cmp rn, rm`	0	0	1	-	0	0	0	1	-	-	-	0	0
`b rn`	0	1	0	-	-	1	0	0	-	-	-	0	0
`bzs label`	0	1	1	0	0	0	0	0	-	-	-	0	0
`bzs label`	0	1	1	1	1	0	0	0	-	-	-	0	0
`ldr rd, [rn]`	1	0	0	-	0	0	1	0	0	1	1	1	0
`str rs, [#imm3]`	1	0	1	-	0	0	0	0	-	-	0	0	1
`NOP (No Operation)`	1	1	0	-	0	0	0	0	-	-	-	0	0
`mov rd, #imm3`	1	1	1	-	0	0	1	0	1	0	-	0	0

- Os '-' na componente dos dados ($D_{0..7}$) podem assumir o valor 0 ou 1
- Os '-' na componente dos endereços ($A_{0..3}$) determinam o número palavras que vão ter a mesma programação

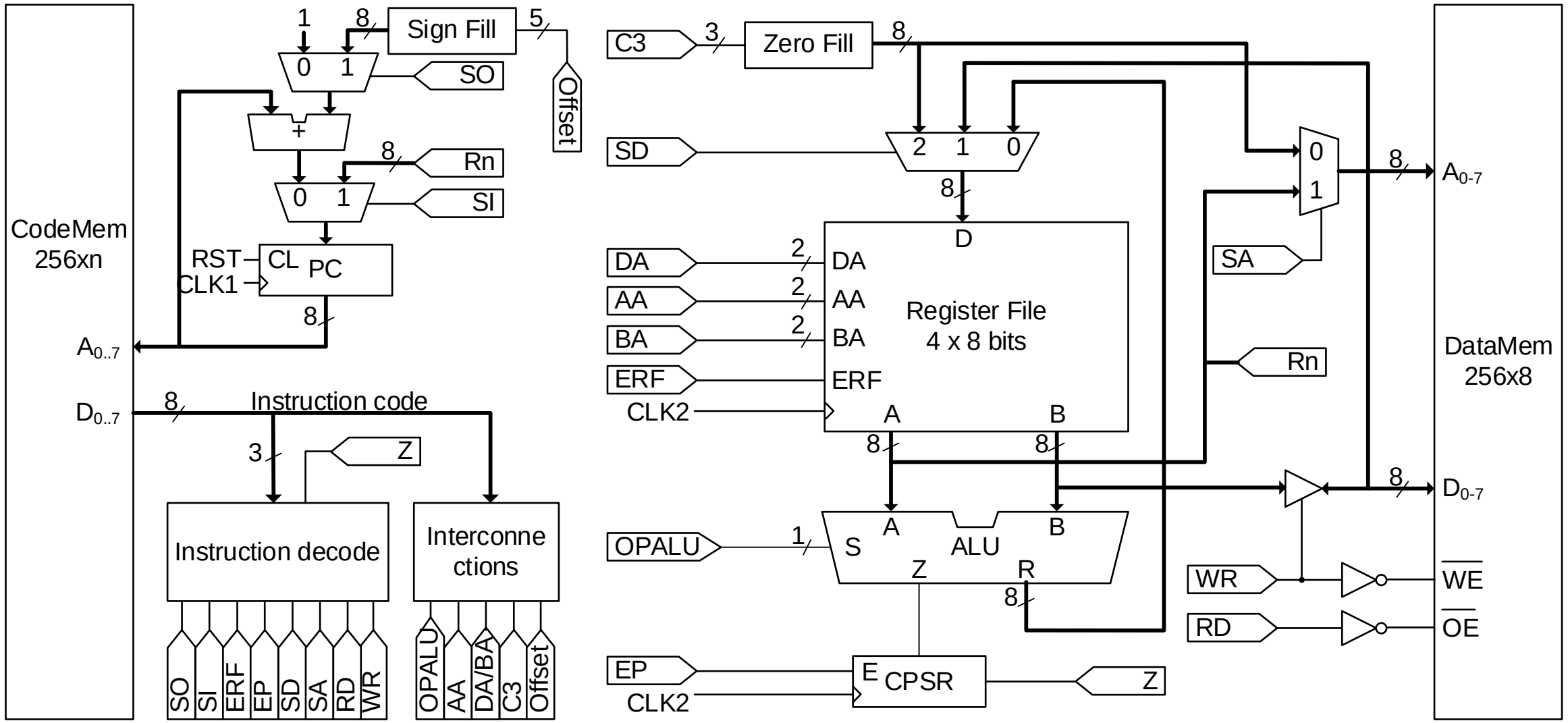
Estrutura interna
Sincronização

- As fontes de *clock* para o registo PC e para os restantes componentes (*Register File* e CPSR) devem ser síncronos mas em oposição de fase para garantir estabilidade dos dados à entrada dos módulo funcionais no momento do registo





Estrutura interna

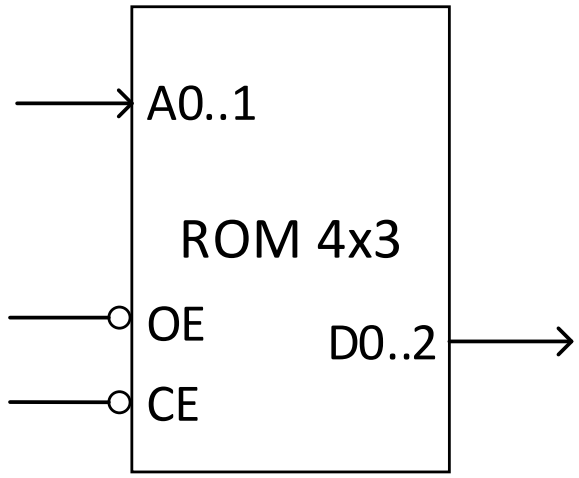


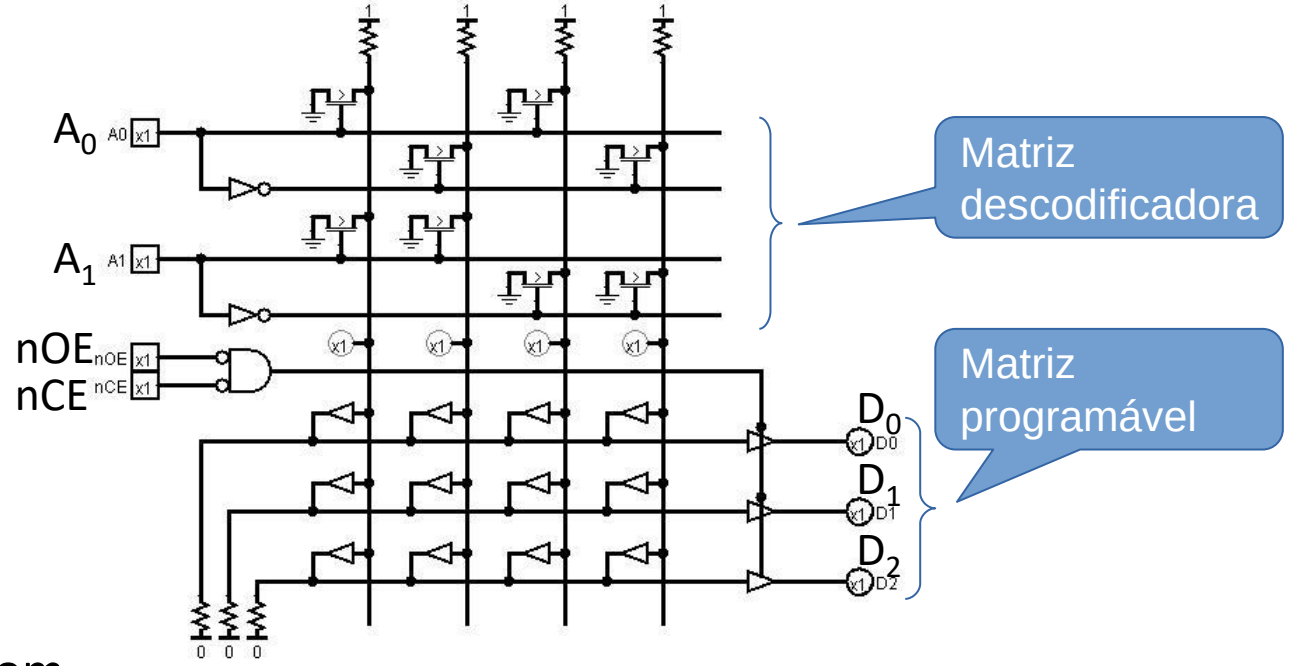
Memória ROM (*Read Only Memory*)

- Memória não volátil: os dados armazenados permanecem válidos mesmo depois de se desligar a alimentação
- Dispositivo endereçável constituído por N palavras de M bits que permite apenas acessos de leitura
  - O número de bits do barramento de endereços determinam o número de palavras
  - O número de bits do barramento de dados determinam o número de bits de cada palavra
- Vários tipos de ROMs:
  - PROM (*Programmable ROM*)
  - EPROM (*Erasable Programmable ROM*)
  - EEPROM ou E2PROM (*Eletrically Erasable Programmable ROM*)
  - *Flash Memory*
- Uma ROM pode ser usada com dois propósitos:
  - Armazenar dados: cada palavra armazena M bits de dados
  - Gerar funções combinatórias:
    - cada bit do barramento de dados representa uma função
    - cada bit do barramento de endereços representa uma variável da função

Memória ROM
Interface externa e estrutura interna

- Exemplo da interface externa de uma ROM 4x3 bits e respetiva estrutura interna:

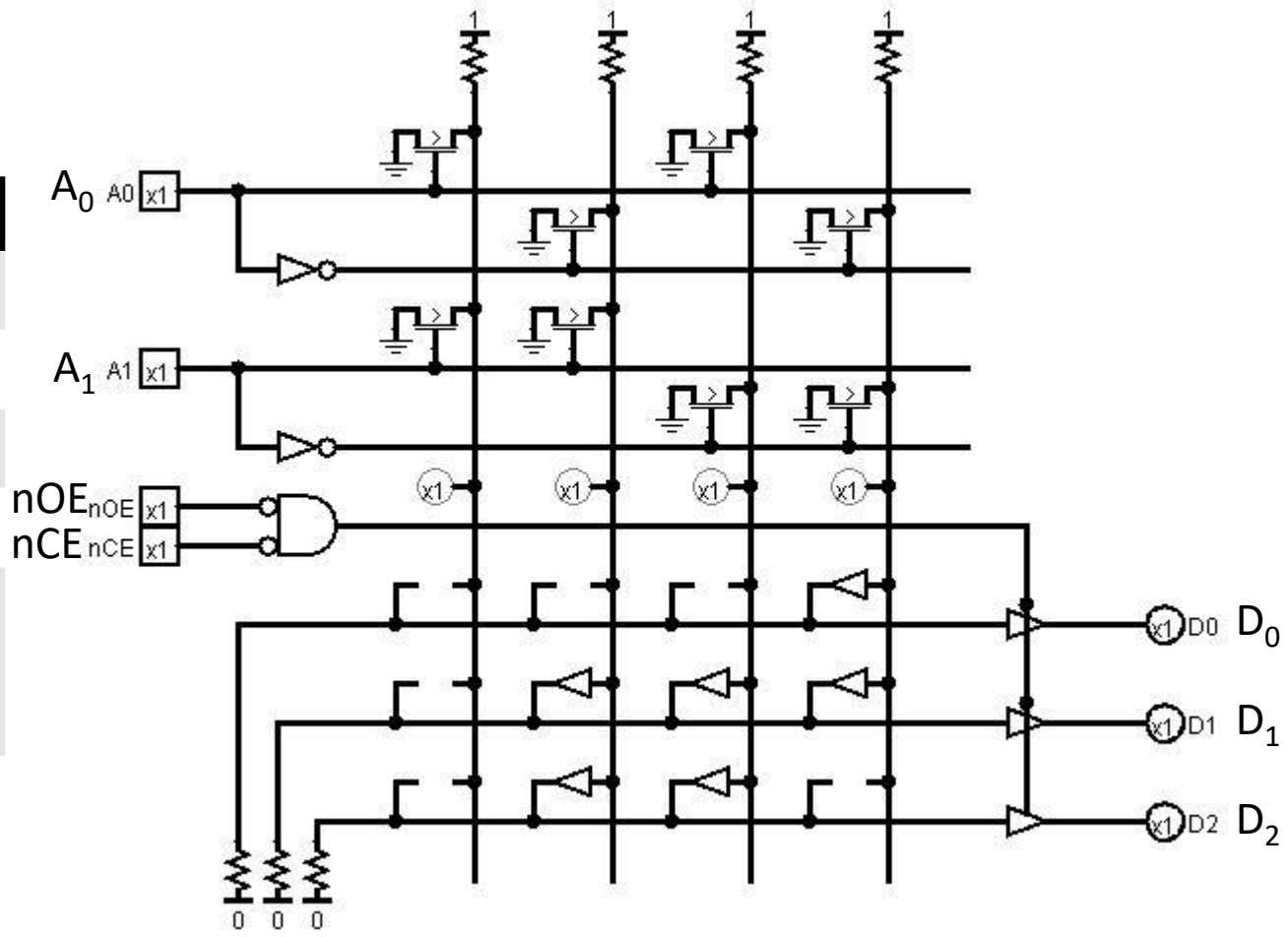




- Matriz descodificadora
  - Não programável/permanece sempre com as mesmas ligações
  - Apenas uma das linhas verticais apresenta o valor lógico 1, função de A0 e A1
- Matriz programável
  - A figura apresenta a matriz de uma ROM não programada

Memória ROM
Exemplo de uma matriz programável programada

A1	A0	D2	D1	D0	Word
0	0	0	0	0	ROM[0] = 0
0	1	1	1	0	ROM[1] = 6
1	0	1	1	0	ROM[2] = 6
1	1	0	1	1	ROM[3] = 3
Função		XOR	OR	AND	

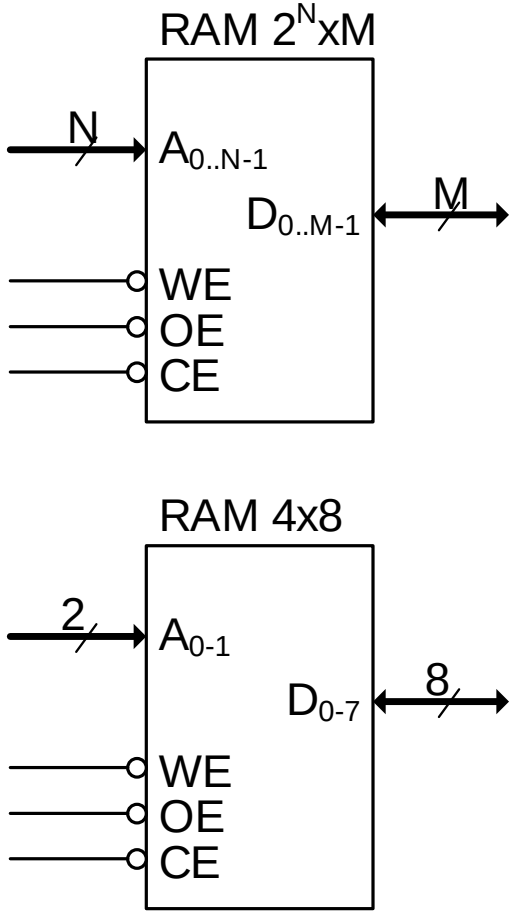


Memória RAM (*Random Access Memory*)

- Memória volátil: os dados armazenados perdem-se depois de se desligar a alimentação
- Dois tipos de RAMs:
  - Estáticas: na hierarquia de memória da arquitetura PC implementam as *caches* (à data, na ordem dos Megas)
  - Dinâmicas: na hierarquia de memória da arquitetura PC implementam a memória principal (à data, na ordem dos Gigas)
  - Internamente, o elemento memória é implementado de forma distinta que as tornas distintas segundo as seguintes características:
    - Tempos de acesso: RAM estática mais rápida que RAM dinâmica;
    - Densidade: na mesma área consegue-se implementar mais bits de memória na RAM dinâmica do que na RAM estática;
    - Custo: as RAMs estáticas são mais caras que as RAMs dinâmicas
- O estudo é focado nas RAMs estáticas

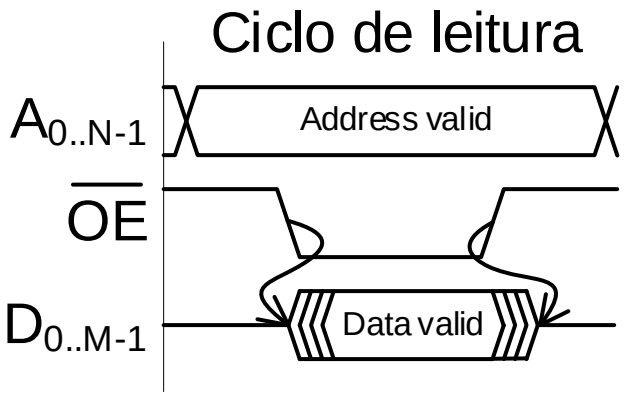
Memória RAM
Interface externa

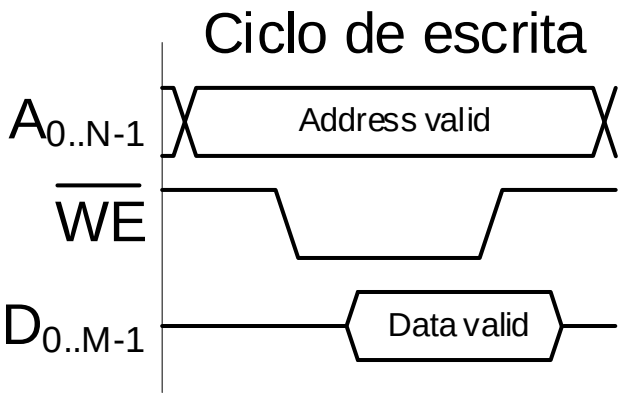
- Sinais de controlo: leitura (RD ou OE); escrita (WR ou WE) e seleção (CS ou CE)
- Barramento de endereços (A): o número de bits deste barramento (N) define o número de registos da memória (2^N)
- Barramento de dados (D): o número de bits deste barramento (M) define o número de bits de cada registo (M)
- Tipicamente, barramento de dados bidirecional



Memória RAM (*Random Access Memory*)
Tipos de acesso

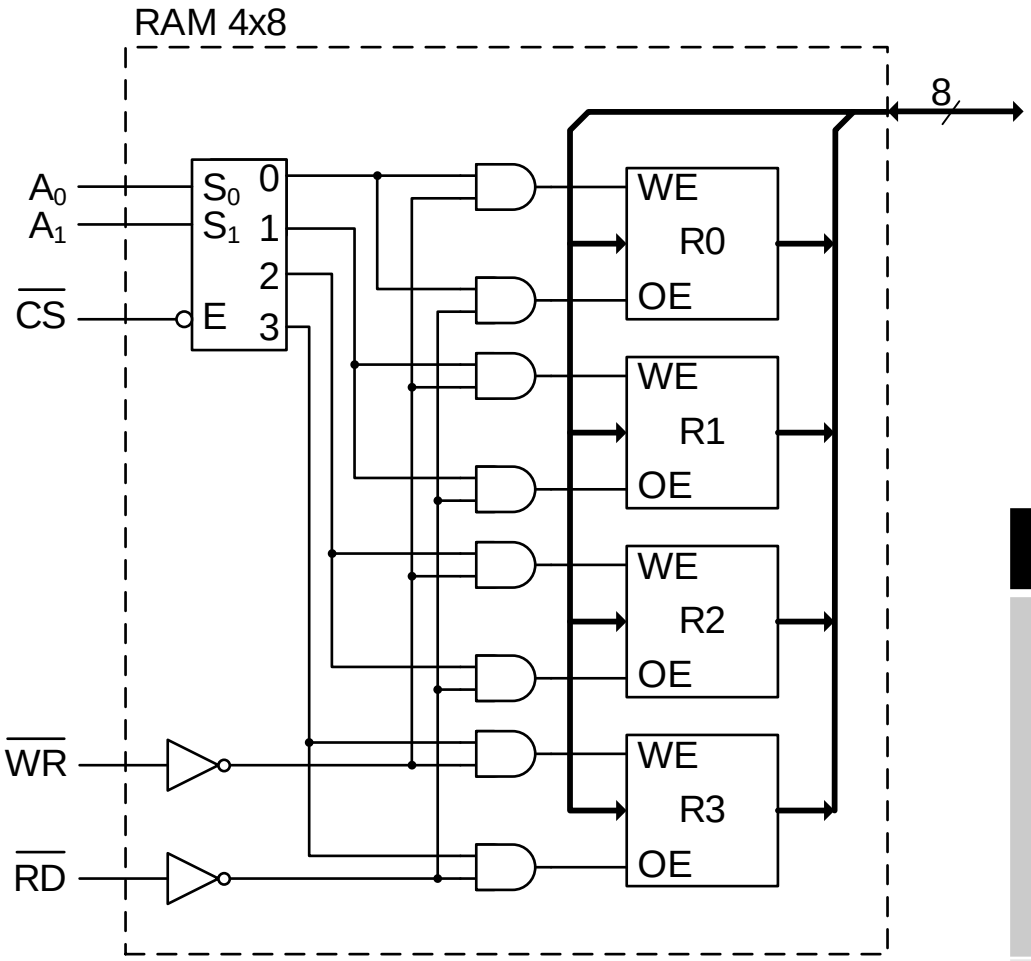
- Dois tipos de acesso:
  - Acesso de leitura: a memória coloca no barramento de dados D o valor do registo selecionado pelo endereço estabelecido no barramento A
  - Acesso de escrita: a memória escreve no registo selecionado pelo endereço estabelecido no barramento A o valor presente no barramento de dados D

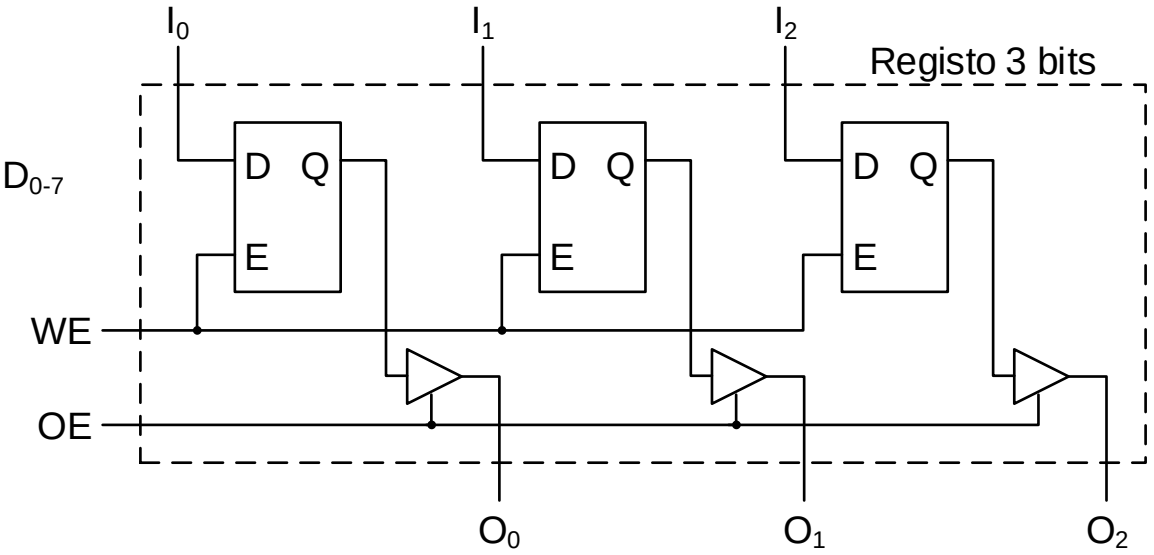




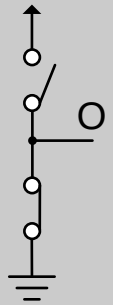
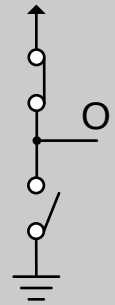
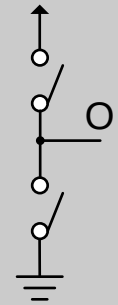
Memória RAM (*Random Access Memory*)

Estrutura interna para RAM 4x8 (4 registos de 8 bits cada)





Andar de saída de porta tri-state

		
Baixa impedância S=0	Baixa impedância S=1	Alta impedância S=z

Programa exemplo

```
ADDR BC Instruction
0000 E0 mov r0, #0
0001 E5 mov r1, #1
0002 F6 mov r2, #5
0003 83 ldr r3, [r0]
0004 07 add r3, r1
0005 2E cmp r3, r2
0006 62 bzs L0
0007 A3 str r3, [#0]
0008 40 L0: b   r0
```

	7	6	5	4	3	2	1	0
1. add rd, rn	0	0	0	-	rn		rd	
2. b rn	0	1	0	-	rn		-	-
3. bzs label	0	1	1	label				
4. cmp rn, rm	0	0	1	-	rn		rm	
5. ldr rd, [rn]	1	0	0	-	rn		rd	
6. mov rd, #imm3	1	1	1	imm3			rd	
7. str rs, [#imm3]	1	0	1	imm3			rs	

Programa exemplo em memória

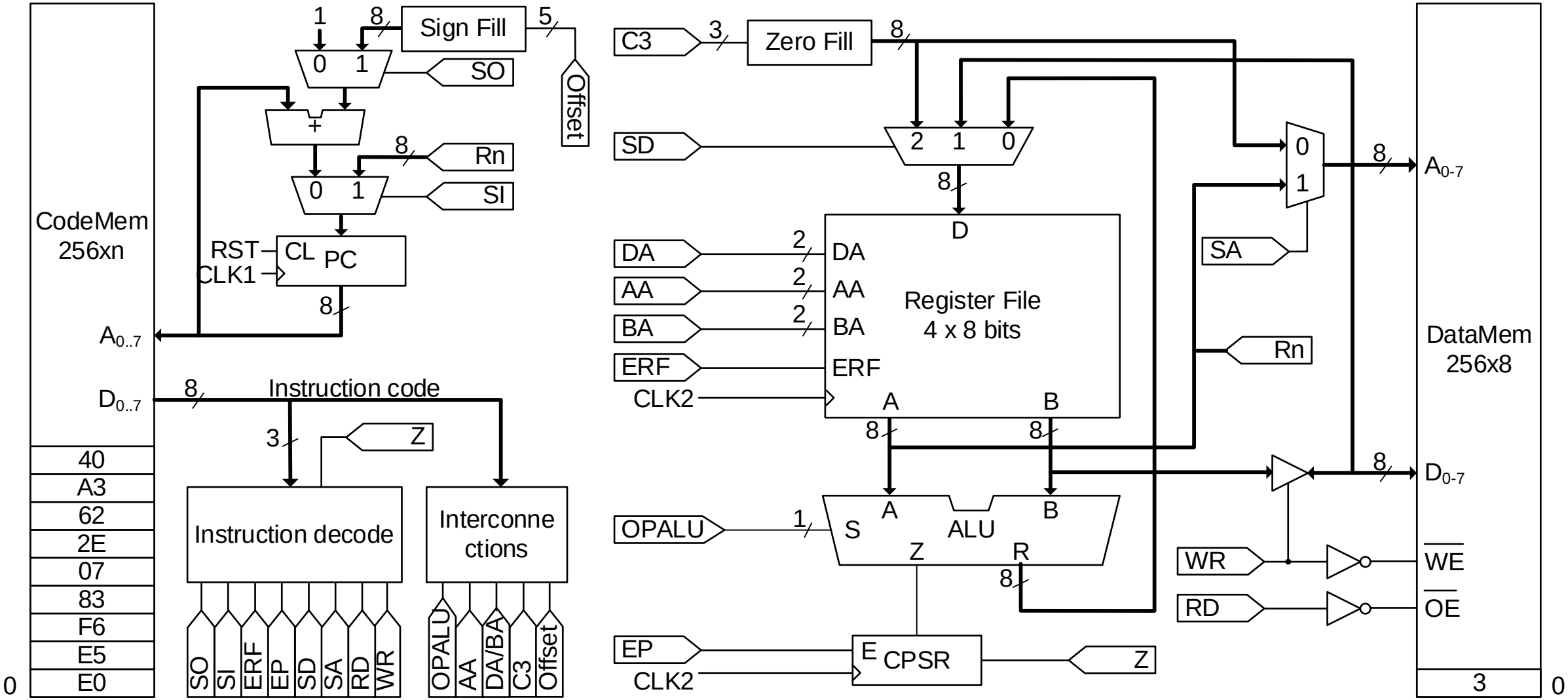
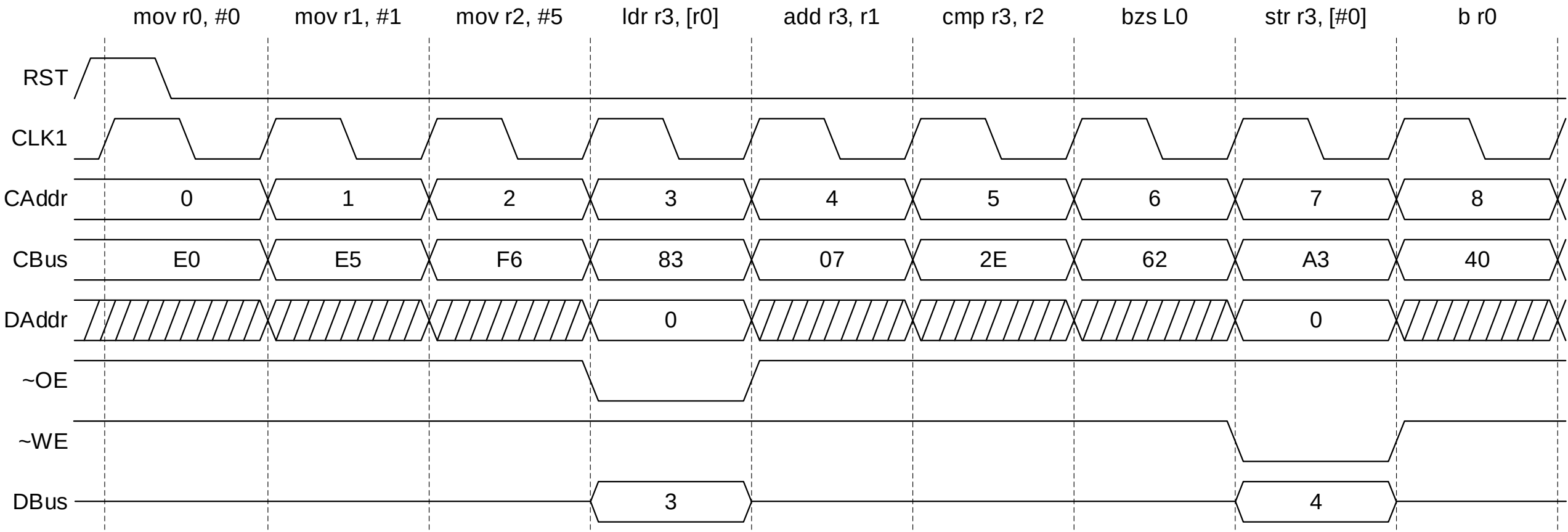


Diagrama temporal produzido pela execução do programa exemplo



- Arquitetura de ciclo único: cada instrução demora um ciclo de *clock* para ser executada