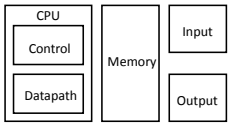
O Processador (CPU)

António de Brito Ferrari
ferrari@ua.pt

AC I – 2ª parte

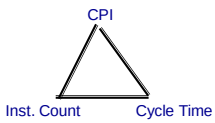
- Os 5 componentes de um computador



- Tema da segunda parte do programa de AC I:
  a estrutura interna do processador

Nível de desempenho do CPU

- Performance determinada por:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction



- Desenho do Processador datapath e control) determina:
  - Clock cycle time
  - Clock cycles per instruction
- O desenho mais simples: **Single cycle processor**
  - Todas as instruções executadas num ciclo de relógio (CPI = 1)
  - Disvantagem: cycle time longo (relógio lento)

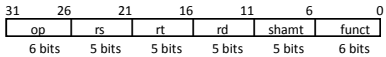
Fases da conceção de um processador

1. Analisar o instruction set => datapath requirements
   - O significado de cada instrução é dado pelas *transferências entre registos*
   - datapath tem de incluir hardware para os registos do ISA
   - datapath tem de suportar as transferências entre registos
2. Selecionar os components para o datapath e definir a metodologia para os impulsos de relógio
3. Construir o datapath de modo a satisfazer as especificações
4. Analisar a implementação de cada instrução para identificar os sinais de controlo que acionam as transferências entre registos
5. Realizar a lógica de controlo

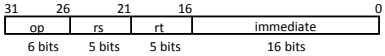
1. Analisar o instruction set

Análise do instruction set

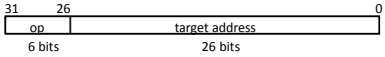
- 3 formatos de instrução:
  - R-type



  - I-type



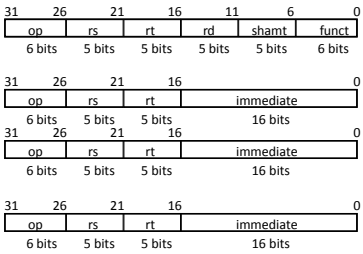
  - J-type



  - op: operação da instrução
  - rs, rt, rd: especificam os registos source e destino
  - shamt: shift amount
  - funct: extensão do código de operação
  - address / immediate: address offset ou immediate value
  - target address: target address do jump

Instruções a implementar

ADD and SUB
addU rd, rs, rt
subU rd, rs, rt
OR Immediate
ori rt, rs, imm16
LOAD e STORE
lw rt, rs, imm16
sw rt, rs, imm16
BRANCH
beq rs, rt, imm16



Requesitos do IS

- Memória: instruções e dados
- Registos (32 x 32)
  - read RS
  - read RT
  - Write RT ou RD
- PC
- Sign Extender
- Add e Sub registos ou extended immediate
- Add 4 ou extended immediate ao PC

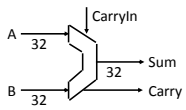
2. Selecionar os componentes para o datapath e método de *clocking*

Componentes do Datapath

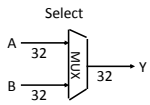
- Elementos combinatórios
- Componentes para armazenar informação (registos)
  - Relógio

Elementos combinatórios

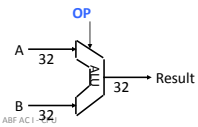
Somador



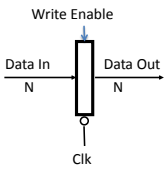
Multiplexer



ALU



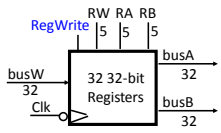
Elementos sequenciais: Registo



Registo: N Flip-Flops
Negative edge-triggered
Write Enable
- 0: Data Out não muda
- 1: Data Out = Data In na próxima frente ativa do relógio

Register File

Register File: 32 registos
2 output buses de 32 bits
busA e busB
1 input bus de 32 bits
busW

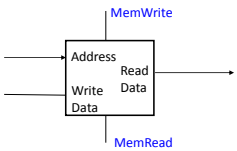


Seleção dos registos:
RA seleciona o registo cujo conteúdo é colocado no busA
RB seleciona o registo cujo conteúdo é colocado no busB
RW seleciona o registo onde é escrito o dado no busW quando RegWrite = 1

Memória

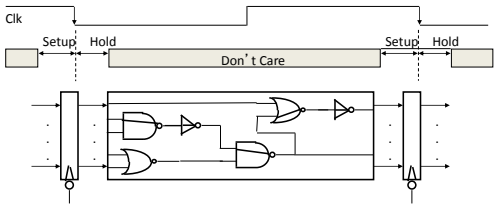
1 Address bus
1 Input data bus (Write Data
1 Output bus (Read Data)

2 sinais de controlo:
MemRead
MemWrite



Seleção da palavra de memória:
MemRead = 1: Address seleciona a palavra a colocar em Read Data
MemWrite = 1: Address seleciona a palavra de memória onde é escrito o dado presente em Write Data

Clocking



Todos os elementos de memória são sensíveis à mesma frente de relógio
Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
(CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

3. Construção do datapath

RTL

- RTL - Register Transfer Logic – expressa o significado das instruções

inst	Register Transfers	
ADDU	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUBU	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00 else PC <– PC + 4	

Ciclo de Instrução

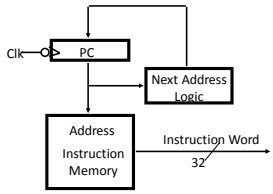
1. Instruction Fetch – PC endereça a memória; leitura do código de instrução;

   Atualização do PC

   IR <– MEM[PC] ;

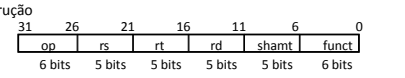
   PC <– PC + 4 (instruções de Branch e Jump – outro valor para o PC)
2. Instruction Execution – a operação especificada no código de instrução é executada

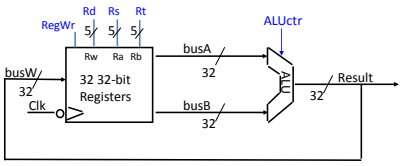
1. Instruction Fetch



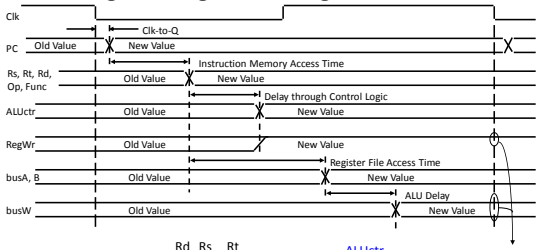
2.1. Execute Type R: Add & Subtract

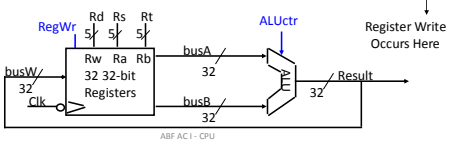
- R[rd] <- R[rs] **op** R[rt] Exemplo: addU rd, rs, rt
  - Ra, Rb, Rw: campos rs, rt, e rd do código de instrução
  - ALUctr, RegWr: gerados pela lógica de controle depois de decodificar a instrução





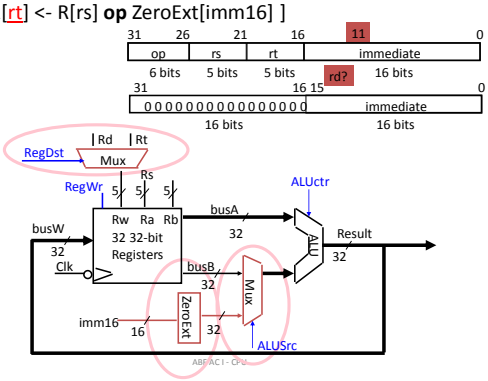
Register-Register Timing





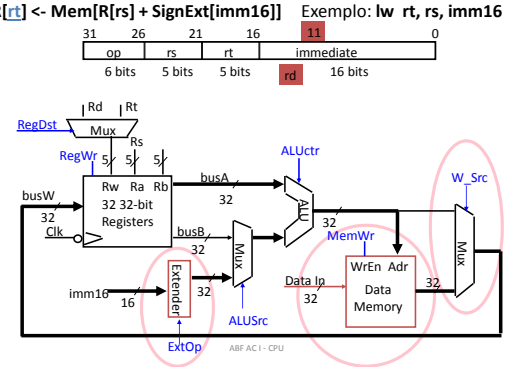
2.2. Execute Type I: a. Operações com Immediate

- R[rt] <- R[rs] **op** ZeroExt[imm16]]



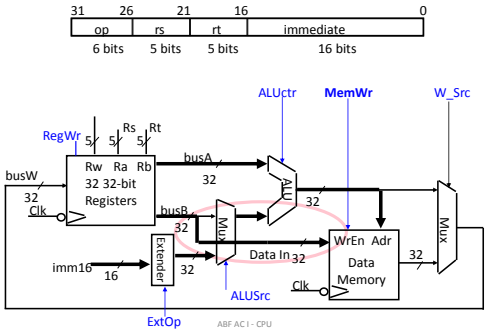
2.2. Execute Type I: b. Load Operations

R[rt] <- Mem[R[rs] + SignExt[imm16]] Exemplo: **lw rt, rs, imm16**

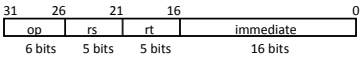


2.2. Execute Type I: c. Store Operations

Mem[R[rs] + SignExt[imm16]] <- R[rt] Exemplo: **sw rt, rs, imm16**



2.2. Execute Type I: d. Branch



- **beq rs, rt, imm16**

Mem[PC] # Fetch da instrução

Equal <- R[rs] == R[rt] # Calcular a condição de branch

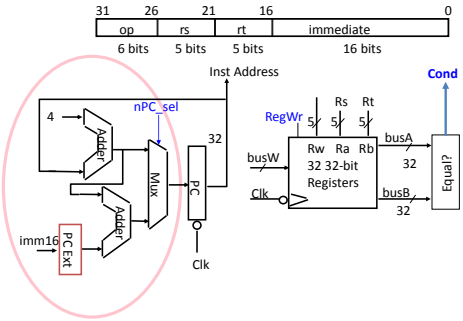
if (COND eq 0) # Calcular endereço da instrução seguinte

PC <- PC + 4 + (SignExt(imm16) x 4)

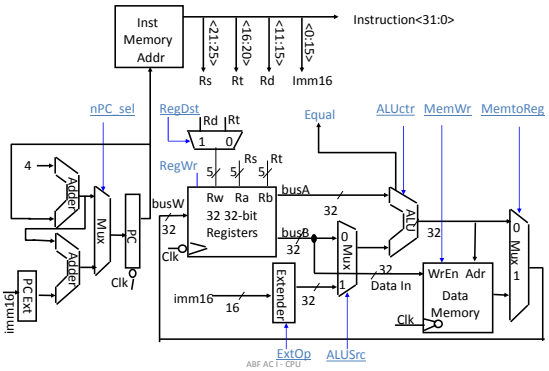
else PC <- PC + 4

2.2. Execute Type I: d. Branch - Datapath

beq rs, rt, imm16 Datapath gera condição (igual)



Single Cycle Datapath Completo



Critical Path

- Register file e memória:
  - CLK input é um fator SÓ durante Memory Write
  - Durante Read, a memória comporta-se como lógica combinatória:
    - Address valid => Output valid after "access time."

Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew