EA075
Processadores de Propósito Geral

Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (UNICAMP)

Prof. Rafael Ferrari

(Documento baseado nas notas de aula do Prof. Levy Boccato)

Introdução

- **Processador de propósito geral ou genérico:** sistema digital programável projetado para resolver tarefas de computação em uma ampla gama de aplicações.

- Exemplos:
  - ARM 7
  - Motorola 68HC05 e 68HC11
  - Intel 8051
  - Intel i3,i5 e i7
  - Atmel ATMega328P (Arduino)

Motivação

- Um projetista de sistemas embarcados pode escolher utilizar um processador genérico para implementar parte de uma funcionalidade desejada do sistema e, com isso, obter alguns benefícios:
  - O custo (de aquisição) por unidade do processador pode ser baixo – o NRE foi amortizado pela grande quantidade (milhões ou até mesmo bilhões) de unidades vendidas.
  - O fabricante pode investir um alto capital em NRE durante o projeto do processador sem que isto aumente de forma significativa o custo da unidade – logo, pode recorrer a tecnologias mais avançadas de IC (e.g., VLSI *layouts*) para componentes críticos.
    - Por isso, processadores genéricos podem oferecer bom desempenho, bem como tamanho e consumo de potência aceitáveis.
  - O custo NRE do projetista é relativamente baixo: basta preparar um *software* e utilizar compiladores / montadores adequados.
  - Tempo de prototipagem e tempo para o mercado são relativamente baixos.
  - Alta flexibilidade.

Arquitetura básica

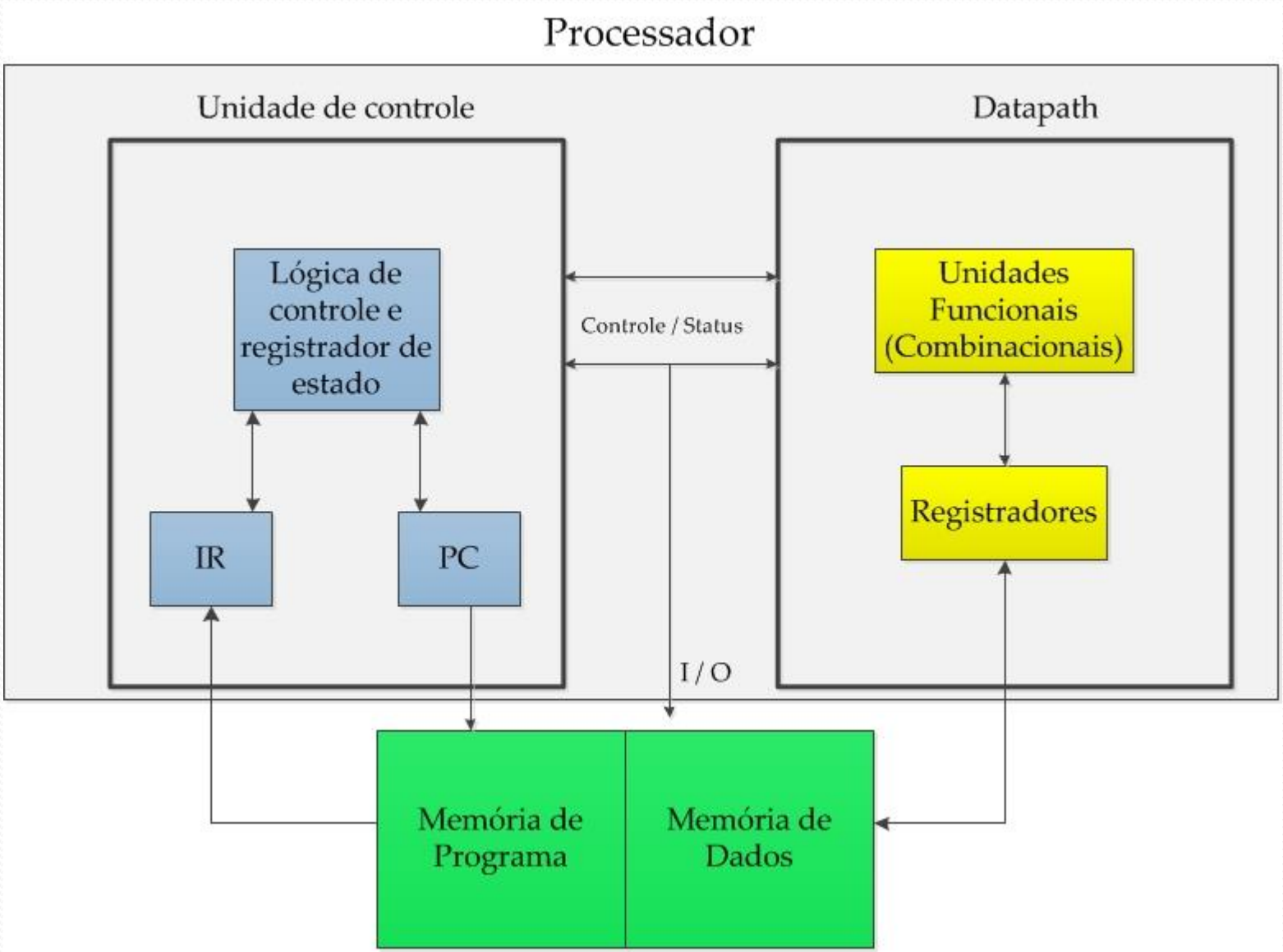
- Processador de propósito geral (CPU, *central processing unit*):
  - *Datapath*
  - Unidade de controle
  - Memória

Semelhante ao processador dedicado, exceto:

(1) pelo fato de o *datapath* ser genérico, oferecendo uma coleção de operações gerais sobre dados;

(2) por ter uma unidade de controle que não realiza uma sequência pré-definida de comandos (precisa ler as instruções armazenadas em uma memória).

Arquitetura básica



Arquitetura básica

- ***Datapath*:**
  - Unidade lógico-aritmética (ULA) – oferece um conjunto de transformações sobre os dados, como adição, subtração, AND, OR, inversão e deslocamento.
  - Gera sinais de *status* que indicam condições particulares referentes às operações executadas (por exemplo, estouro aritmético (*overflow*), adição que gera um vai-1 (*carry*)).
  - Registradores para armazenamento temporário de dados.
  - Operação *load*: transfere o conteúdo de uma posição de memória para um registrador.
  - Operação *store*: transfere o conteúdo de um registrador para uma posição de memória.
  - Define o tamanho do processador (regist. de *N* bits, operações executadas sobre operandos de *N* bits, barramentos, interfaces de dados).

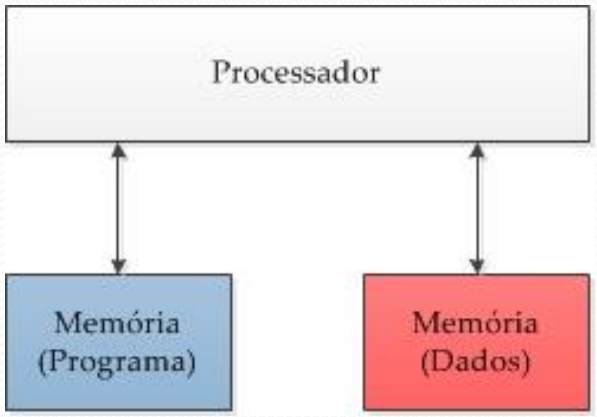
Arquitetura básica

- **Unidade de controle:**
  - Circuito que sequencia a execução de instruções de programa, sendo responsável por mover os dados de, para e através do *datapath* de acordo com estas instruções.
  - **Registrador PC:** contém o endereço da próxima instrução a ser lida.
    - O controlador ajusta o valor do PC para sempre apontar para a próxima instrução. No caso de um desvio ou de uma ramificação, sinais de status do *datapath* podem nortear a definição do próximo valor de PC.
    - Seu tamanho determina o espaço de endereçamento do processador: por exemplo, se PC tem 16 bits, existem 65536 posições de memória endereçáveis.
  - **Registrador IR:** contém a instrução lida.
  - Cada instrução exige que o controlador passe por vários estágios, sendo que cada estágio pode durar um ou mais ciclos de relógio.
  - A frequência do processador dá uma noção de sua velocidade.

Arquitetura básica

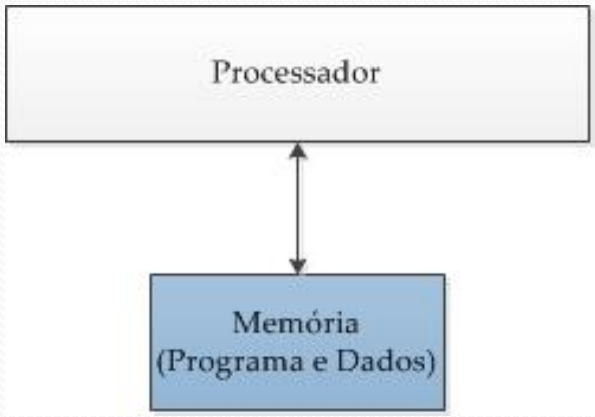
- **Memória:**
  - Armazenamento de dados e instruções para médio e longo prazos.
  - **Duas arquiteturas:**

Harvard



Leitura simultânea de dados e de instruções

Princeton



Menor quantidade de conexões

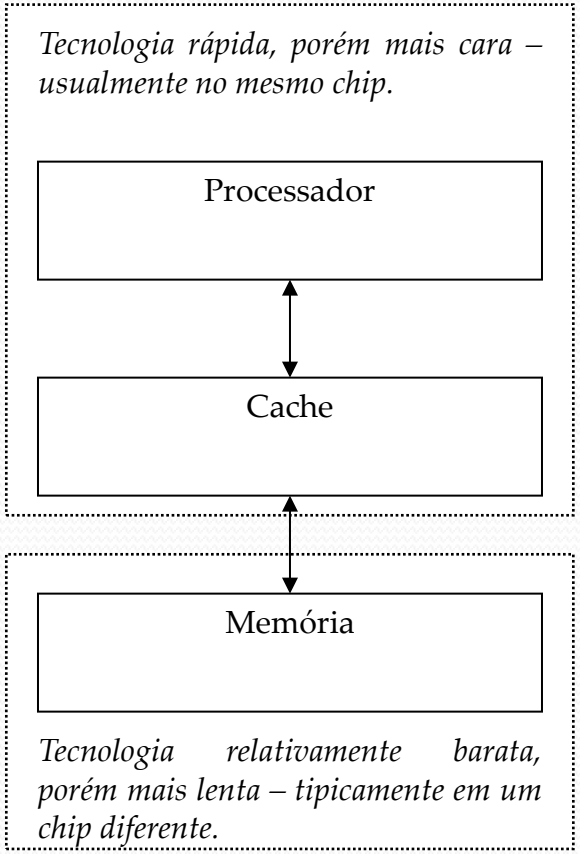
Arquitetura básica

- **Memória:**
  - Existem diferentes tipos de memória, por exemplo, ROM e RAM.
  - On-chip: a memória está no mesmo CI que o processador.
    - Acesso mais rápido, porém com maiores limitações de capacidade (tamanho).
  - Off-chip: memória está em um CI separado.

Arquitetura básica

- **Memória:**

Para reduzir o tempo de acesso à memória, uma cópia local (no mesmo chip do processador) de parte da memória é mantida em um pequeno, mas especialmente rápido, dispositivo chamado de *cache*.



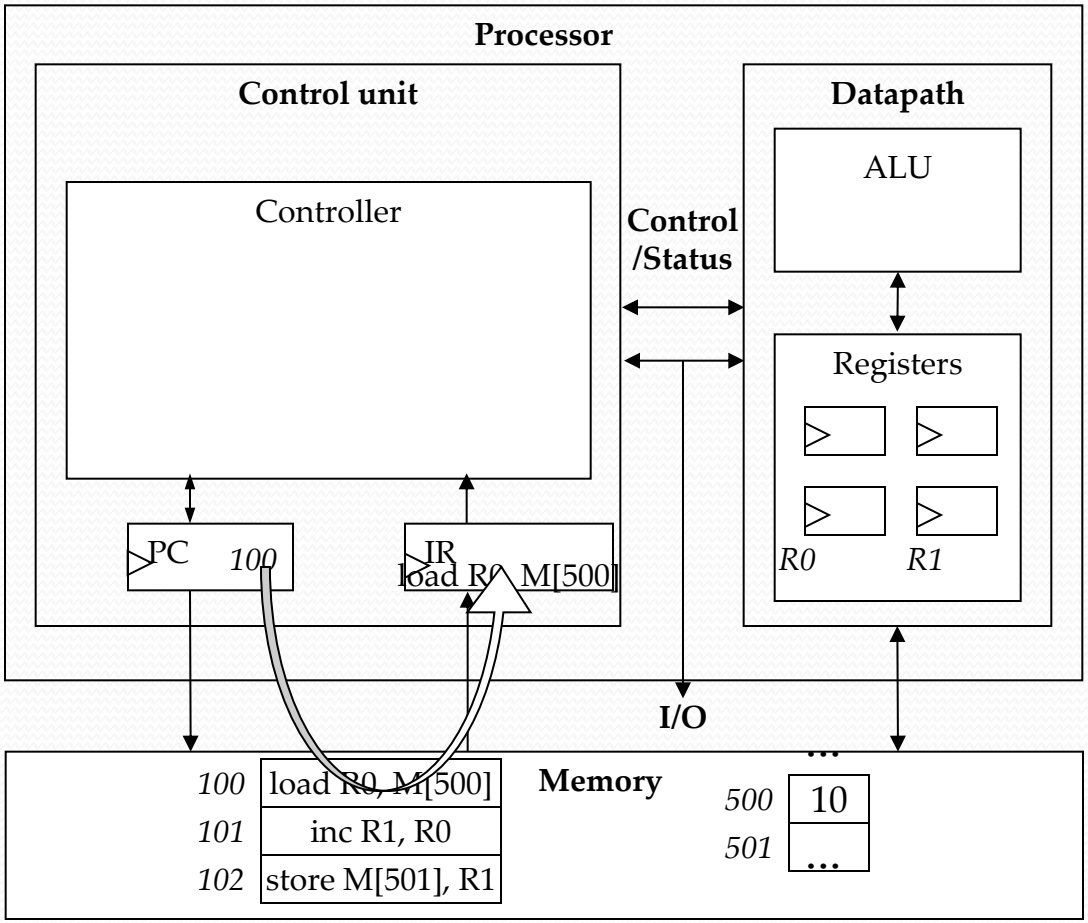
Operação

- **Execução de uma instrução**

  - Busca (*fetch*) de instrução.
  - Decodificação.
  - Busca de operandos.
  - Execução da operação.
  - Armazenamento de resultados.

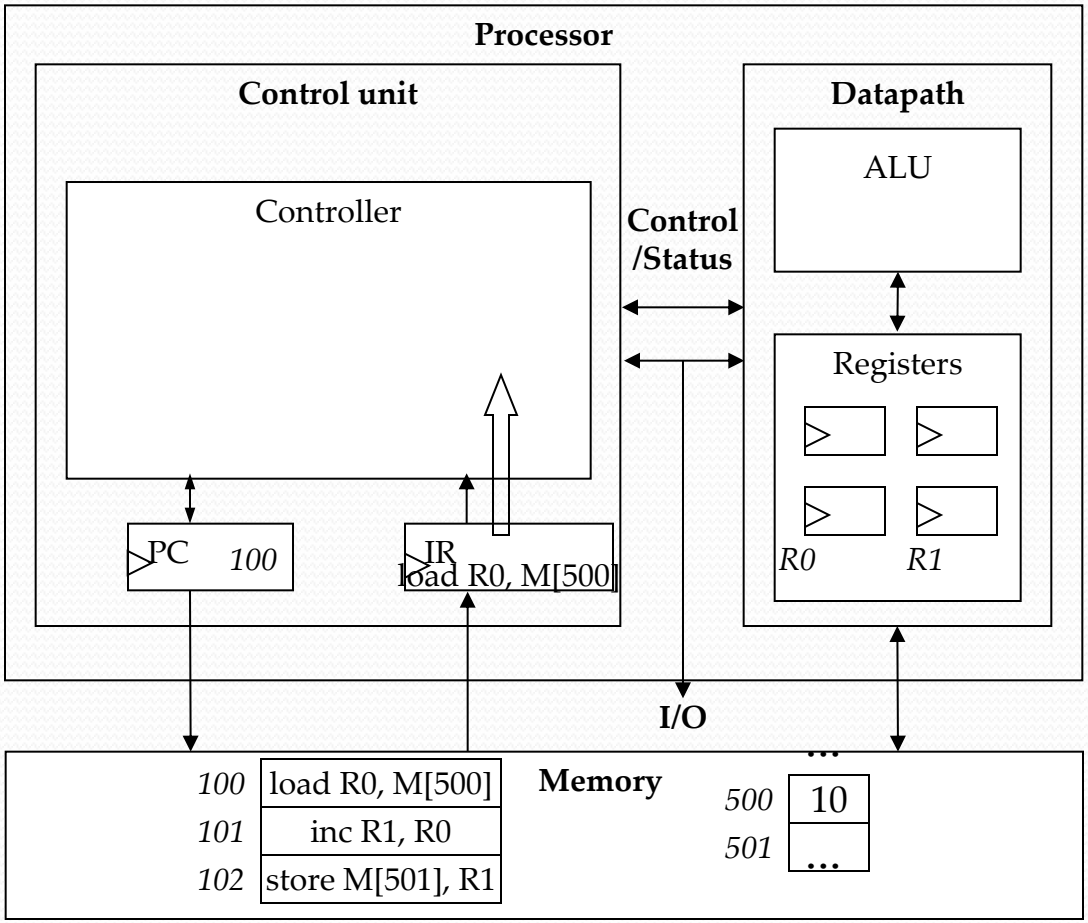
Operação

- Busca de instrução (BI):
  - Carrega a próxima instrução no IR (registrador de instrução).
  - PC (Program Counter), sempre aponta para a próxima instrução.
  - IR: armazena a instrução a ser executada.



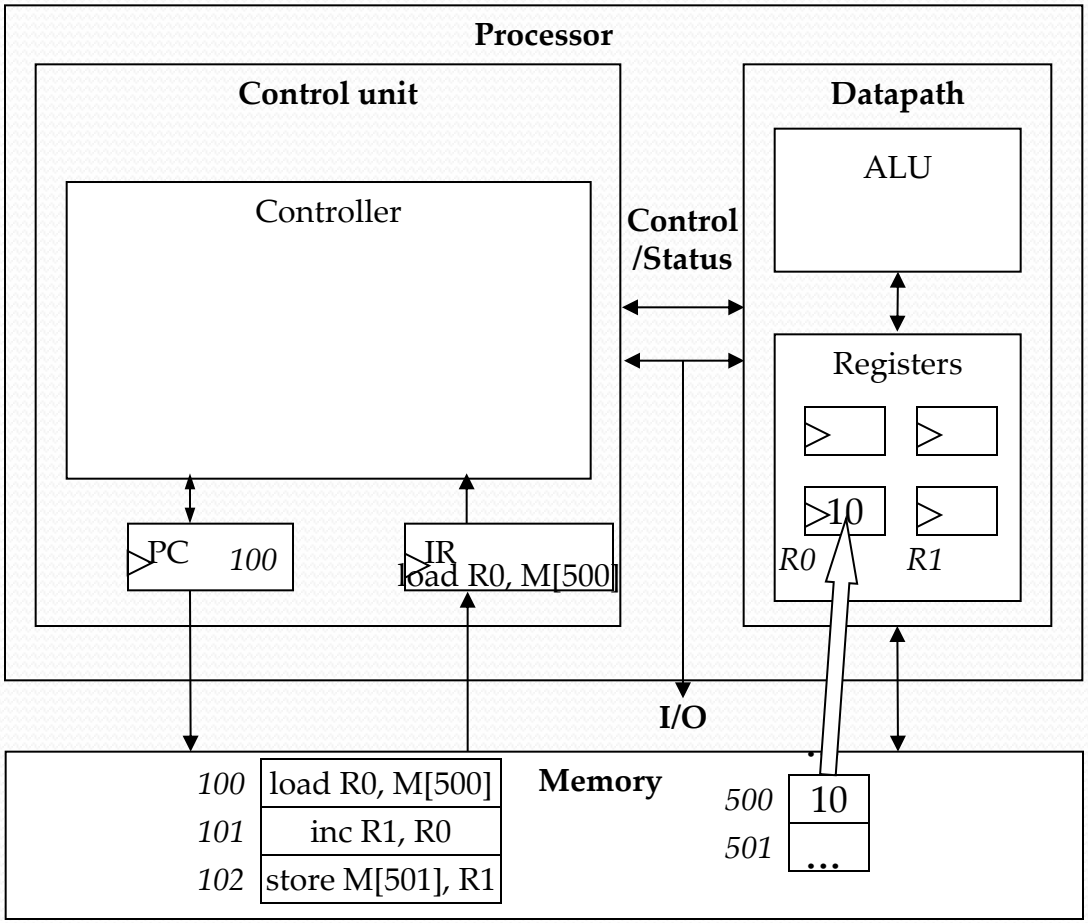
Operação

- Decodificação de instrução (DI):
  - Determina qual operação está especificada na instrução armazenada no IR.



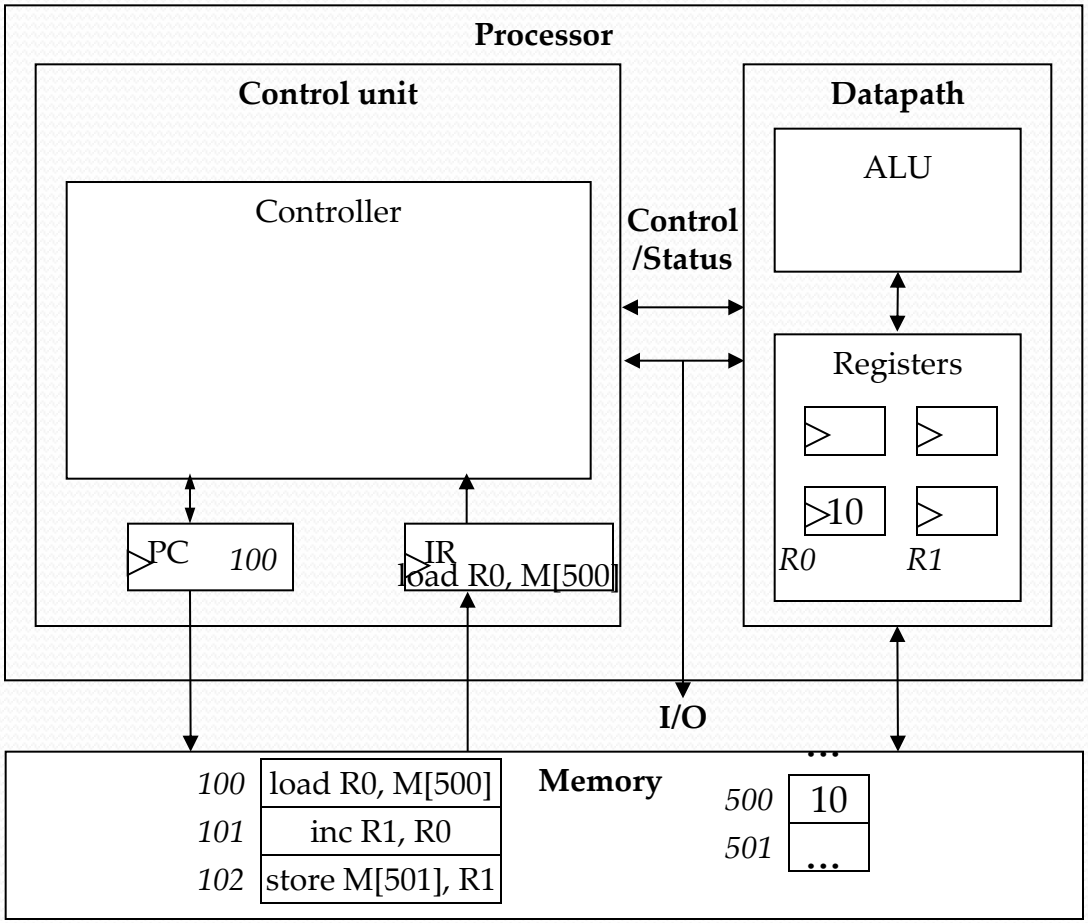
Operação

- Busca de operandos (BO):
  - Movimentação dos operandos da instrução para os registradores apropriados.



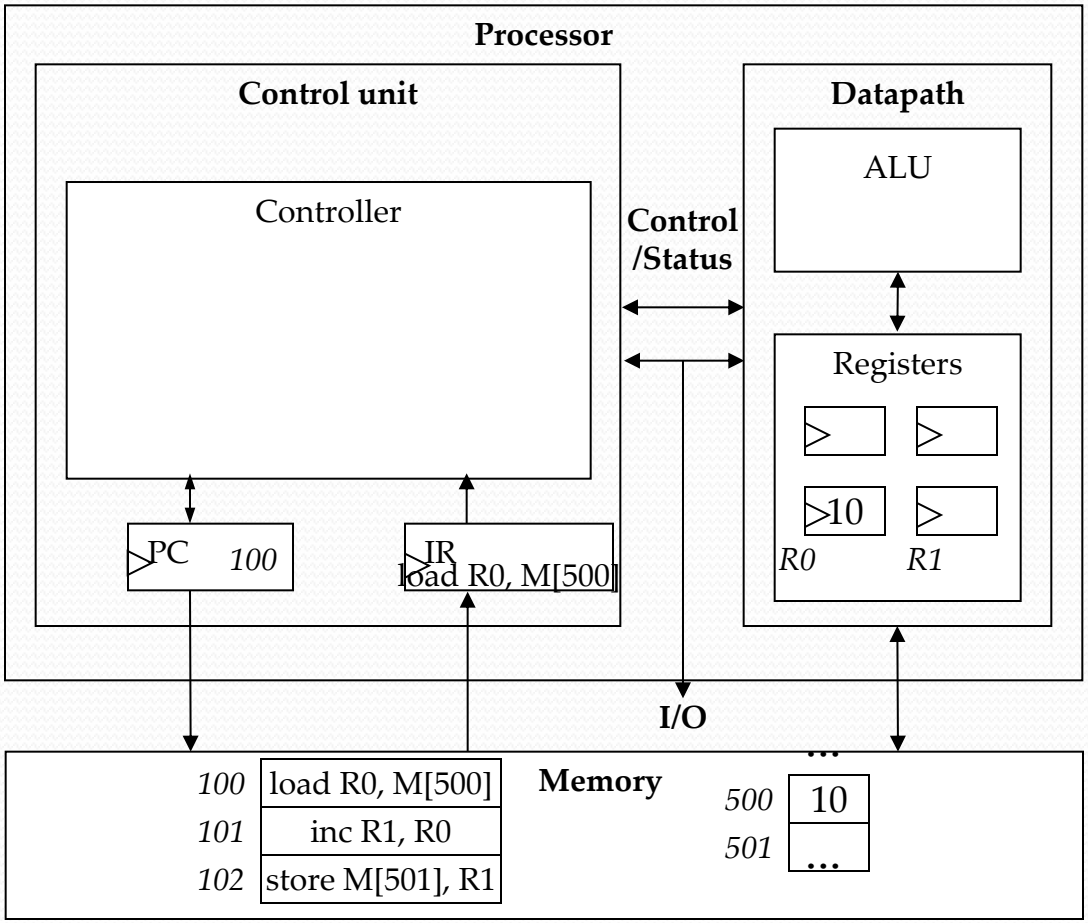
Operação

- Execução de instrução (EI):
  - Alimenta os componentes apropriados da ALU que realizarão a operação desejada.
  - A instrução *load* não faz uso da ALU e portanto não realiza nenhuma tarefa durante esta sub-operação.



Operação

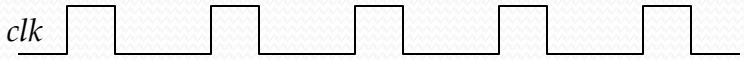
- Armazenar resultados (AR):
  - Escrita de resultados armazenados em registradores de volta à memória.
  - A instrução *load* não realiza nenhuma tarefa durante esta sub-operação.

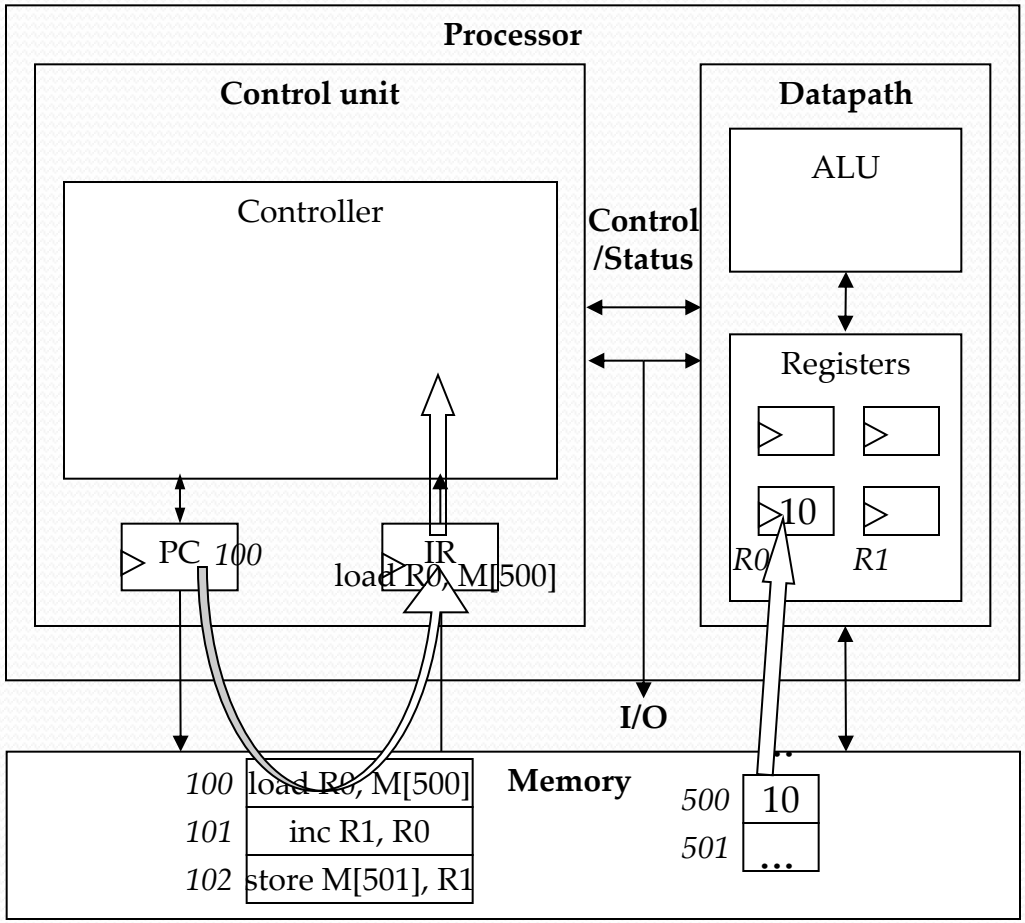


Ciclos de Instrução

PC=100

BI DI BO EI AR

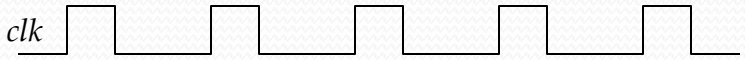




Ciclos de Instrução

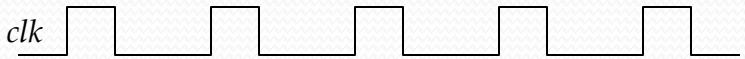
PC=100

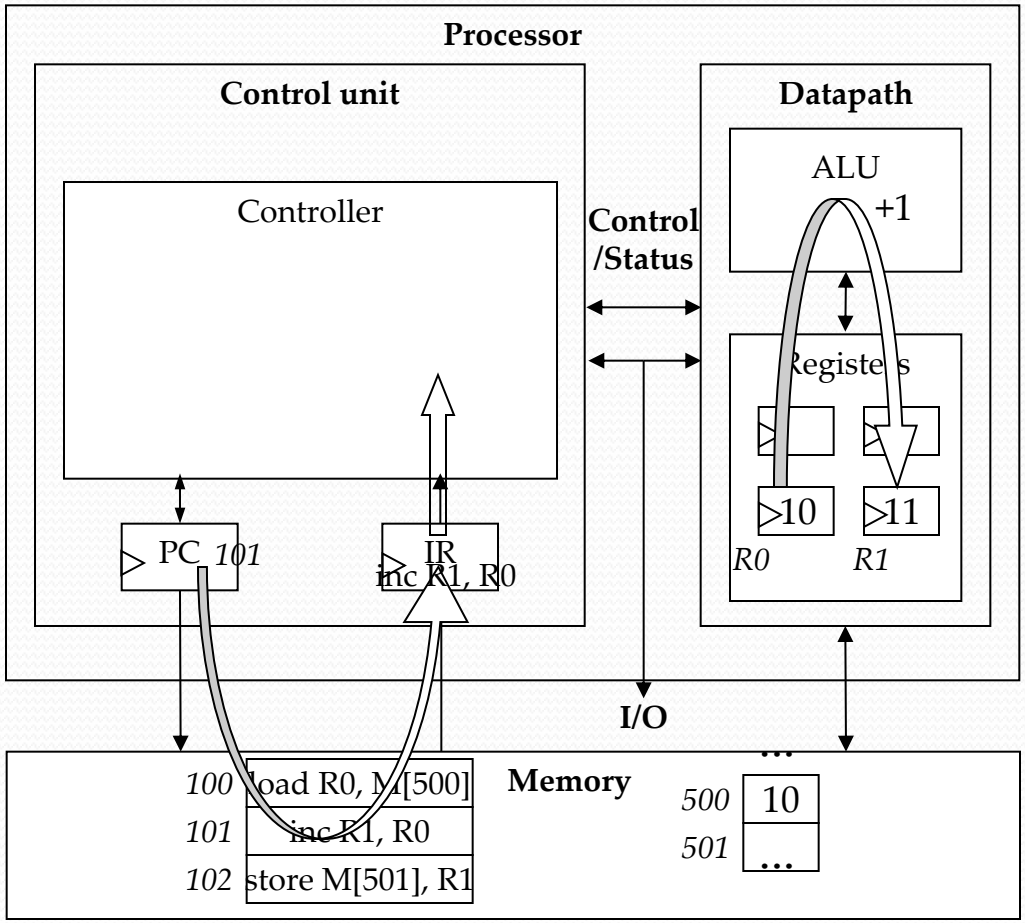
BI DI BO EI AR



PC=101

BI DI BO EI AR





Operação

PC=100

BI DI BO EI AR

clk

PC=101

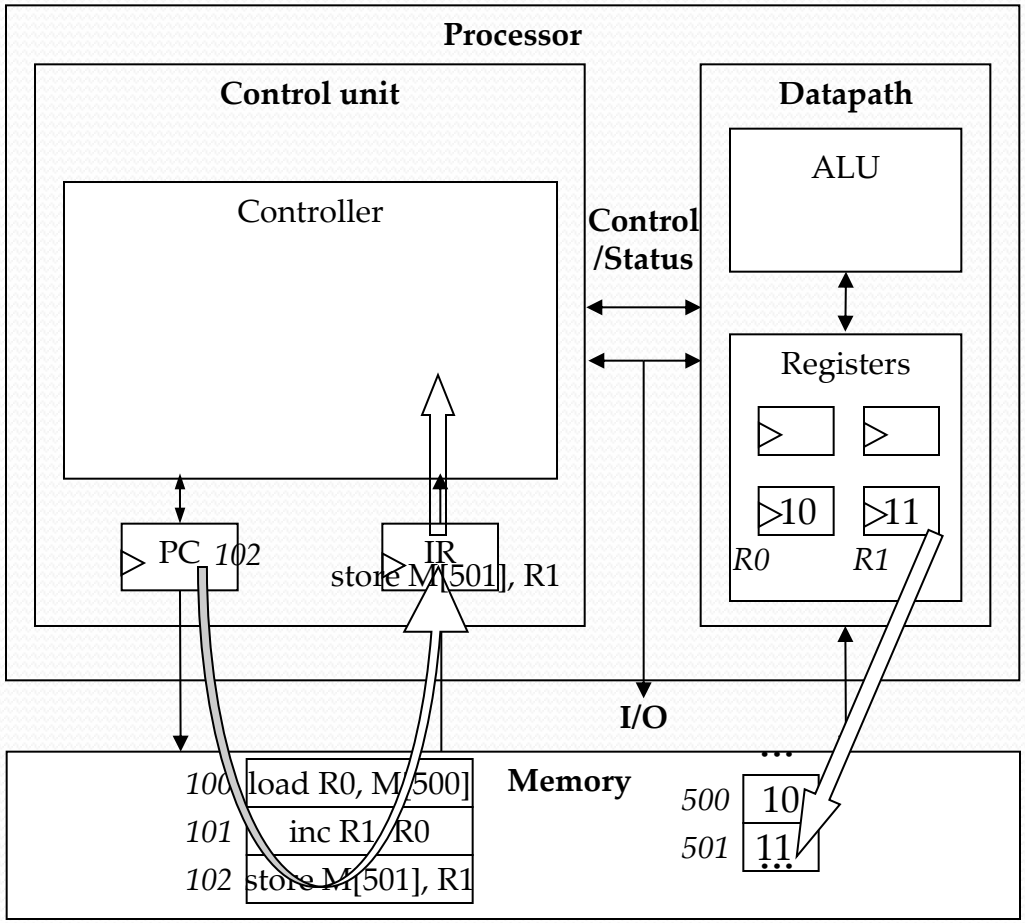
BI DI BO EI AR

clk

PC=102

BI DI BO EI AR

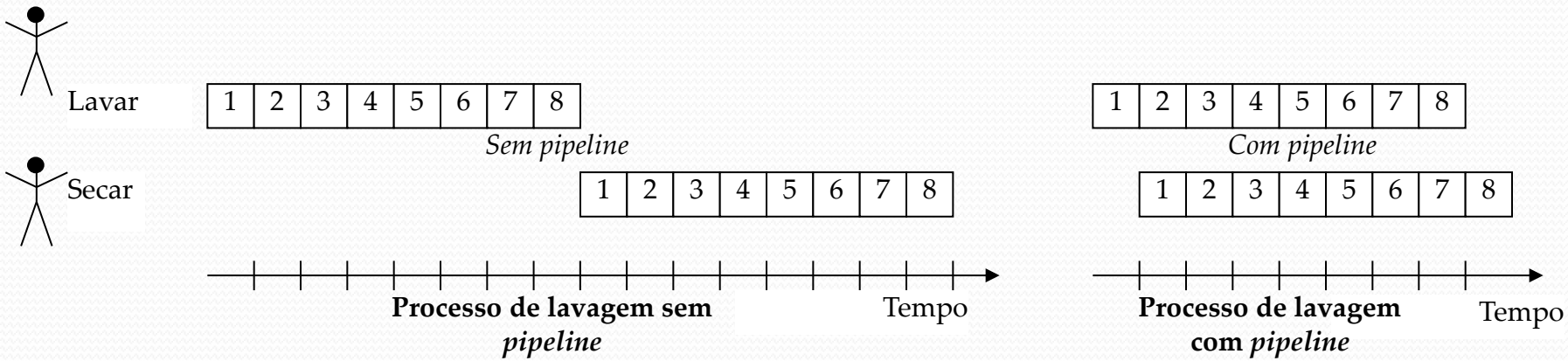
clk



Pipeline

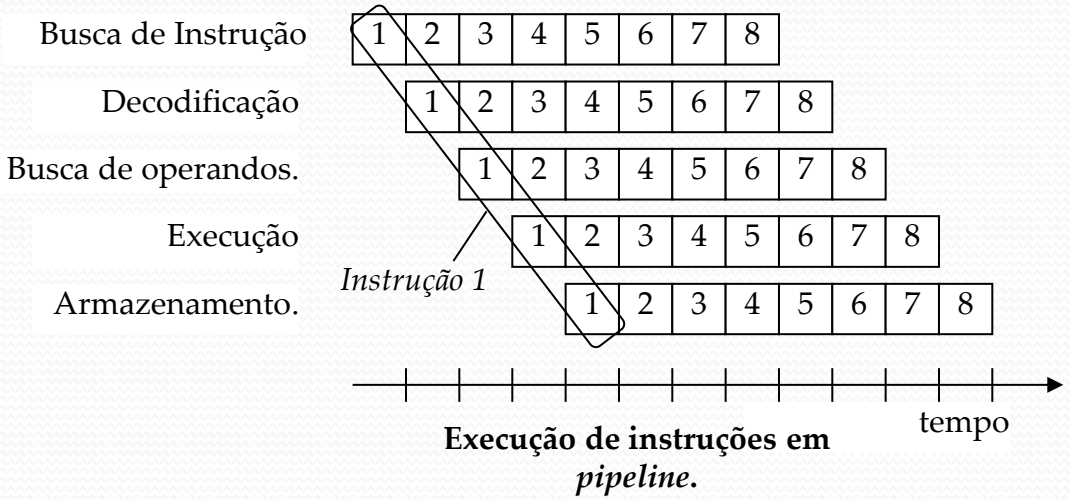
- ***Pipeline*:** ao utilizar uma unidade separada para cada estágio de instrução, é possível ter múltiplas instruções de máquina em processamento ao mesmo tempo.

- Ilustração: lavar louças.



Pipeline

- Exemplo: depois que a unidade de busca de instrução (*fetch*) faz a leitura da primeira instrução, a unidade de decodificação é acionada para interpretá-la, enquanto a unidade de busca pode carregar a próxima instrução em IR.



Execução de instruções em *pipeline*.

Pipeline

- Sem pipeline: 2 instruções em 10 ciclos de relógio

Instrução 1	BI	DI	BO	EI	AR					
Instrução 2						BI	DI	BO	EI	AR
Relógio	1	2	3	4	5	6	7	8	9	10

- Com pipeline: 6 instruções em 10 ciclos de relógio

Instrução 1	BI	DI	BO	EI	AR					
Instrução 2		BI	DI	BO	EI	AR				
Instrução 3			BI	DI	BO	EI	AR			
Instrução 4				BI	DI	BO	EI	AR		
Instrução 5					BI	DI	BO	EI	AR	
Instrução 6						BI	DI	BO	EI	AR
Relógio	1	2	3	4	5	6	7	8	9	10

Desempenho do Pipeline

- Tempo para que um pipeline com k estágios processe n instruções:

$$T_{k,n} = [k + (n - 1)]\tau$$

em que τ é o período de ciclo.

- No exemplo anterior, k = 5 estágios e n = 6 instruções, ou seja, $T_{5,6}$ = [5 + (6 - 1)] τ = 10 τ.
- Em um processador equivalente, com ciclo de instrução dividido em k estágios mas sem pipeline, o tempo para processamento de n instruções é $nk\ \tau$.
- Fator de aceleração do pipeline:

$$S_k = \frac{nk\tau}{[k + (n - 1)]\tau} = \frac{nk}{k + (n - 1)}$$

- Para um número de instruções tendendo a infinito, a aceleração tende a k, ou seja, ao número de estágios do pipeline.
- Nesse sentido, quanto maior o número de estágios, maior a aceleração.
- Entretanto, os ganhos potenciais de estágios adicionais são confrontados pelo aumento da complexidade, do custo e do atraso entre estágios.

Pipeline

- **Cuidados:**

  - As instruções devem ser passíveis de decomposição em estágios de aproximadamente o mesmo tamanho.

  - As instruções devem tomar o mesmo número de ciclos de relógio.

  - Instruções de desvio / ramificação são um problema para *pipelines*, pois não se sabe qual será a próxima instrução até que a atual atinja o estágio de execução.

Pipeline: Hazards

- Há situações especiais em que a próxima instrução do programa não pode ser executada no ciclo seguinte. Estes eventos são chamados de ***hazards***.

- Veremos três tipos:
  A. **Estruturais ou de recursos**
  B. **Dados**
  C. **Controle ou desvio**

Pipeline: Hazards

- ***Hazard* estrutural ou de recursos:**
  - Situação de conflito pelo uso (simultâneo) de um mesmo recurso de *hardware*.
  - Ocorre quando duas instruções precisam utilizar o mesmo componente de *hardware* – para fins distintos ou com dados diferentes – no mesmo ciclo de relógio.
  - **Exemplo:** única memória sendo acessada em dois momentos distintos (e.g., para busca de instrução e para carregamento de um valor em um registrador).

ld R0,M[500]	BI	DI	BO	EI	AR					
Instrução 2		BI	DI	BO	EI	AR				
Instrução 3			Ocioso	BI	DI	BO	EI	AR		
Instrução 4					BI	DI	BO	EI	AR	
Relógio	1	2	3	4	5	6	7	8	9	10

Pipeline: Hazards

- ***Hazard* de dados:**
  - Ocorrem quando uma instrução depende da conclusão de uma instrução prévia que ainda esteja no *pipeline* para realizar sua operação e/ou acessar um dado.
  - **Exemplo:**

    add R1, R2 **; R1 = R1 + R2**

    sub R3, R1 **; R3 = R3 – R1**

    - A instrução *add* somente escreve seu resultado no final do 5º estágio do *pipeline*.
    - Logo, é necessário aguardar dois ciclos de relógio até que o resultado correto (R1) possa ser lido pela instrução seguinte, *sub*.

add R1, R2	BI	DI	BO	EI	AR					
sub R3, R1		BI	DI	**Ocioso**		BO	EI	AR		
Instrução 3			BI			DI	BO	EI	AR	
Instrução 4						BI	DI	BO	EI	AR
Relógio	1	2	3	4	5	6	7	8	9	10

Pipeline: Hazards

- ***Hazard* de controle ou de desvio:**
  - Surge por causa da necessidade de tomar uma decisão baseada em resultados de uma instrução enquanto outras estão em execução.
  - Ou seja, está ligado a instruções de desvio.
  - **Problema:**
    - O *pipeline* inicia a busca da instrução subsequente ao *branch* no próximo ciclo de relógio.
    - Porém, não há como o *pipeline* saber qual é a instrução correta a ser buscada, uma vez que acabou de receber o próprio *branch* da memória.

Pipeline: Hazards

- ***Hazard* de controle ou de desvio:**
  - **Exemplo:** a segunda instrução é um salto para a oitava instrução. Consideramos que o endereço da instrução seguinte ao salto é definido no estágio de execução de instrução (EI).
  - No sexto período do relógio, as instruções 3, 4 e 5 são removidas do pipeline enquanto a instrução 8 é inserida nele.
  - Três ciclos de relógio são perdidos no processo.

Instrução 1	BI	DI	BO	EI	AR						
Salto para a instrução 8		BI	DI	BO	EI	AR					
Instrução 3			BI	DI	BO						
Instrução 4				BI	DI						
Instrução 5					BI						
Instrução 8						BI	DI	BO	EI	AR	
Instrução 9							BI	DI	BO	EI	AR
Relógio	1	2	3	4	5	6	7	8	9	10	11

Operação

- O desempenho de um processador pode ser aperfeiçoado através de :
  - Sinais de relógio mais rápidos (contudo existe um limite).
  - *Pipeline*: divisão da execução das instruções em estágios.
  - Múltiplas ALUs para suportar mais de uma sequência de instruções.

- **Superescalar:** é capaz de realizar duas ou mais operações escalares em paralelo, fazendo uso de duas ou mais ALUs.
  - Estática – a ordem das operações tem que ser definida em tempo de compilação.
  - Dinâmica – reordenam as instruções em tempo de execução para fazer uso de ALUs adicionais.

Visão do programador

- O programador nem sempre precisa conhecer todos os detalhes da arquitetura, podendo trabalhar em diferentes níveis de abstração:
  - Nível Assembly – linguagem comumente ligada às características do processador.
  - Linguagens estruturadas – C, C++, Java, etc.

- A maior parte do desenvolvimento hoje é realizado utilizando-se linguagens estruturadas.
  - Programação em linguagem *Assembly* pode ser necessária.
  - **Drivers:** partes do programa que comunicam com e/ou controlam (dirigem) outros dispositivos.
    - Frequentemente é preciso levar em conta aspectos temporais, manipulação de bits. Nestes casos, *Assembly* pode ser a melhor opção.

Visão do programador

- **Conjunto de instruções:** corresponde ao repertório de operações elementares que o programador pode invocar.

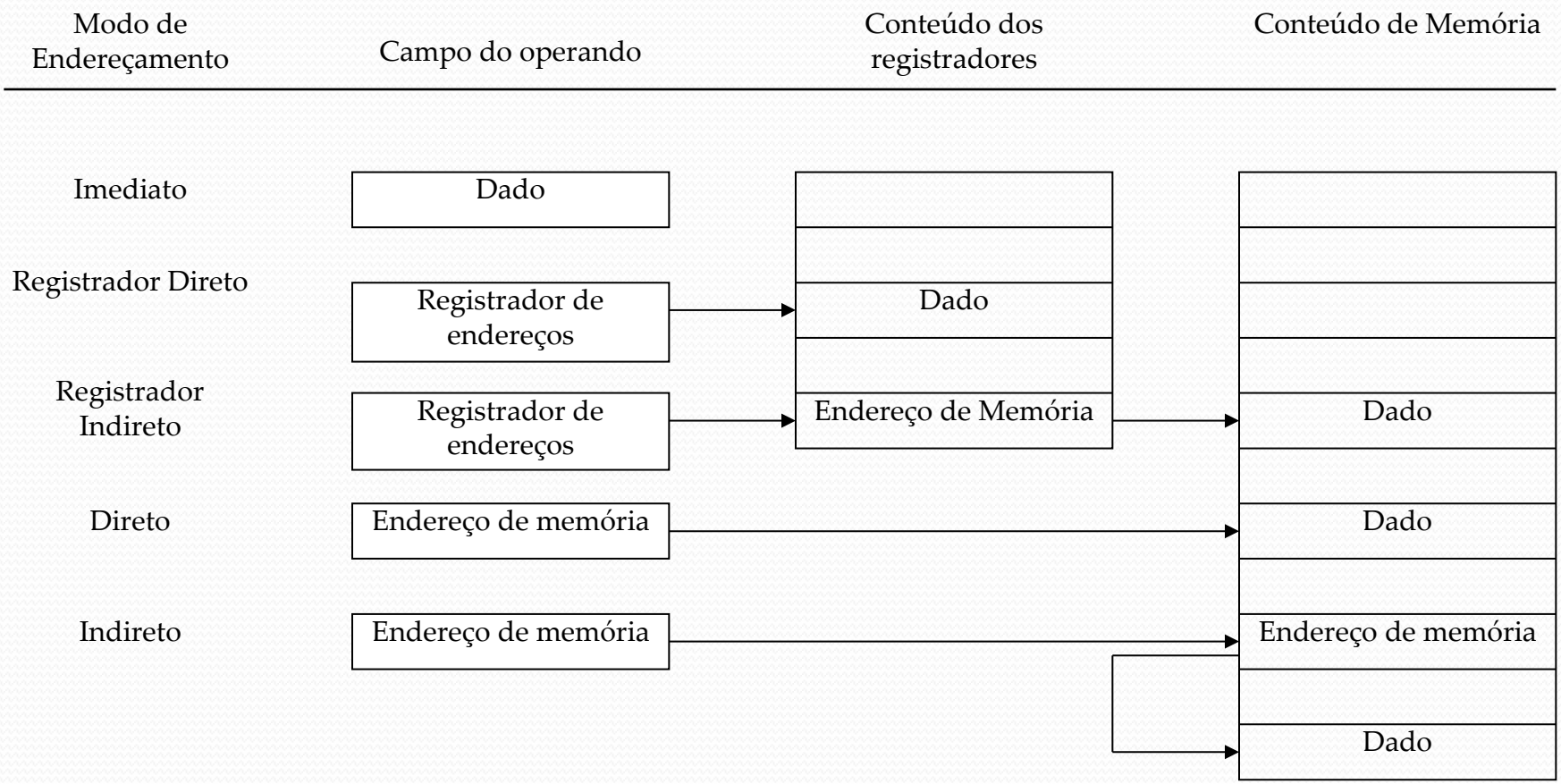
Opcode	1º Operando	2º Operando

  - Opcode: código binário que define um identificador único para cada instrução.

- **Tipos básicos de instruções:**
  - Transferência de dados: memória/registrador, registrador/registrador, I/O, etc.
  - Lógico-aritmética: usa registradores como entradas para a ALU e armazena resultados em registradores.
  - Desvio (*branch*): determina o valor do PC quando deseja-se realizar um salto para outro ponto do código, em vez da próxima instrução imediata.

Visão do programador

- **Modos de endereçamento:**



Visão do programador

- **Espaço para programas e dados**
  - Processadores em sistemas embarcados costumam ser muito limitados: por exemplo, 64 Kbytes de programa, 256 bytes de RAM (expansível).

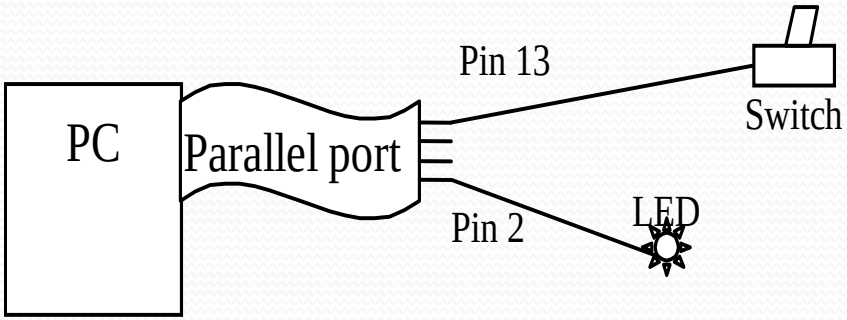
- **Registradores:** quantos estão disponíveis? Existem registradores com funções especiais?
  - Alguns processadores, como os microcontroladores, possuem dispositivos embutidos (timers, ADCs, etc) que são configurados através de registradores específicos.

- **I/O**
  - Como é realizada a comunicação com os dispositivos externos (portas)?

- **Interrupções**
  - Onde deve ser armazenada a rotina de tratamento de interrupção? Quantos pinos do processador são destinados a sinais de interrupção?

Visão do programador

- **Exemplo:** porta paralela monitora a chave de entrada e acende ou apaga o LED de acordo com a posição da chave.



Visão do programador

- Um conjunto especial de três registradores é utilizado para leitura/escrita de valores nos pinos da porta paralela.

Pino de conexão (LPT)	Direção (I/O)	Posição / Registrador
1	Saída	Bit 0 do regist. #2
2-9	Saída	Bits 0-7 do regist. #0
10, 11, 12, 13, 15	Entrada	Bits 6, 7, 5, 4, 3 do regist. #1
14, 16, 17	Saída	Bits 1-3 do regist. #2

- Endereço base do banco de registradores: $3BC_h$.
  - Logo, o regist. #2 está no endereço $3BC + 2 = 3BE_h$.

Visão do programador

- Programa em Assembly

```
CheckPort proc
     push al              ; save the content
     push dx              ; save the content
     mov  dx, 3BCh + 1    ; base + 1 for register #1
     in   al, dx          ; read register #1
     and  al, 10h         ; mask out all but bit # 4
     cmp  al, 0           ; is it 0?
     jne  SwitchOn        ; if not, we turn the LED on

SwitchOff:
     mov  dx, 3BCh + 0    ; base + 0 for register #0
     in   al, dx          ; read the current state of the port
     and  al, FEh         ; clear first bit (masking)
     out  dx, al          ; write it out to the port
     jmp  Done            ; we are done

SwitchOn:
     mov  dx, 3BCh + 0    ; base + 0 for register #0
     in   al, dx          ; read the current state of the port
     or   al, 01h         ; set first bit (masking)
     out  dx, al          ; write it out to the port

Done:
     pop  dx              ; restore the content
     pop  al              ; restore the content
CheckPort endp
```

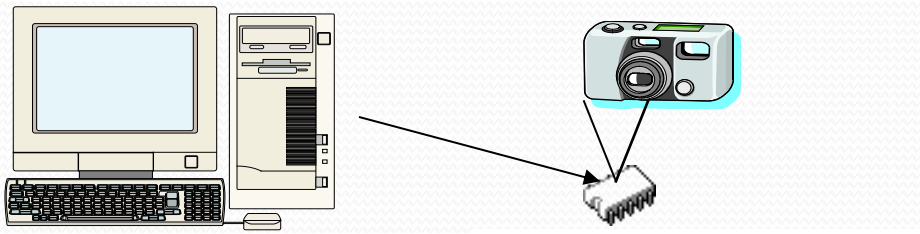
Visão do programador

- Sistema operacional:
  - Esconde alguns detalhes do *hardware* e provê à camada de aplicação (que é onde o programador atua) uma interface para o *hardware* por meio do mecanismo de chamadas de sistema.
  - Administração de arquivos, acesso à memória.
  - Interface com teclado / display e outros dispositivos conectados ao computador.
  - Sequenciamento da execução de múltiplos programas (divisão do tempo de uso da CPU).

Ambiente de Desenvolvimento

- **Processador de desenvolvimento**
  - Corresponde ao processador no qual escrevemos e depuramos o programa.
  - Usualmente presente em um PC.

- **Processador alvo**
  - Corresponde ao processador no qual o programa final será carregado e que irá efetivamente fazer parte da implementação do sistema embarcado.
  - Frequentemente, é diferente do processador de desenvolvimento.



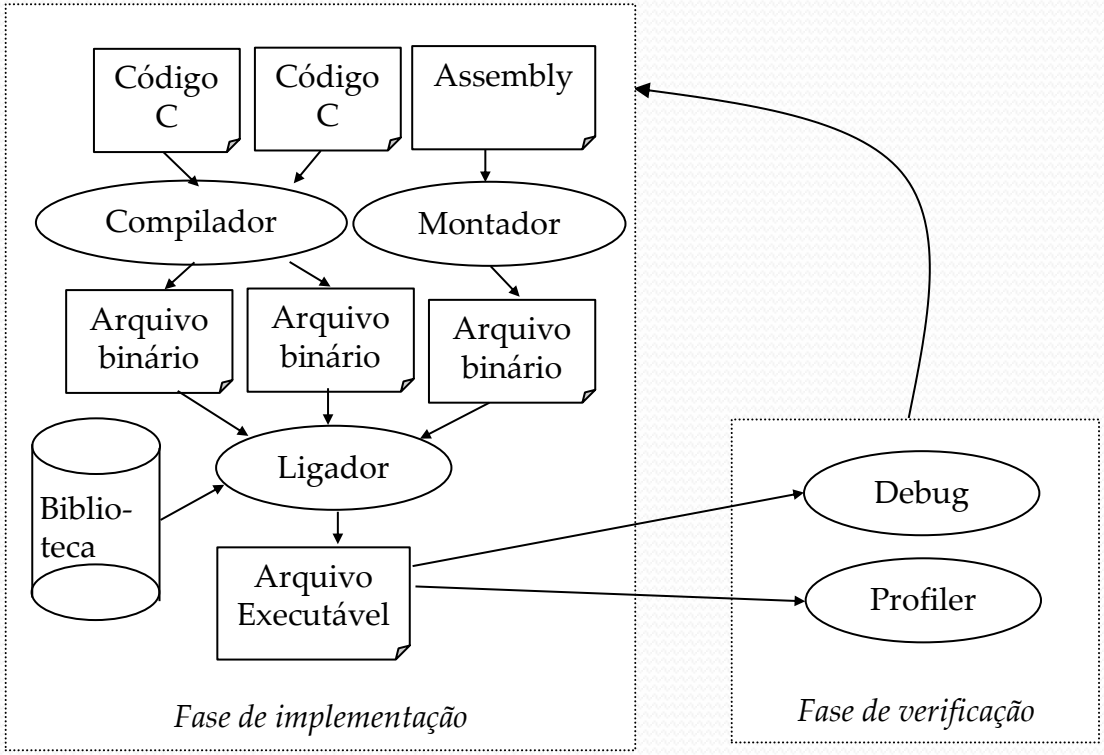
Processador de desenvolvimento

Processador Alvo

Ambiente de Desenvolvimento

- A programação de um processador inserido no sistema embarcado apresenta algumas diferenças sutis, porém importantes, em relação ao projeto de *software* em um *desktop*.

- Em um desktop:



Ambiente de Desenvolvimento

- Em um sistema embarcado, o processador alvo comumente é diferente do processador de desenvolvimento. Logo, embora a programação seja feita no processador de desenvolvimento, o código gerado precisa ser compatível com o formato de instrução utilizado pelo processador alvo.

- **Compiladores:** traduzem programas escritos em linguagens estruturadas em instruções de máquina, possivelmente realizando algumas otimizações no código.
  - **Compilador-cruzado (*cross compiler*):** é executado em um processador (desenvolvimento), mas gera código para outro processador (alvo).

- **Montadores:** traduzem instruções mnemônicas (Assembly) em instruções de máquina (binário), fazendo também a tradução de endereços (no lugar dos rótulos).
  - **Cross-assembler.**

Ambiente de Desenvolvimento

- **Teste e depuração:**
  - Depurar um programa que roda em um sistema embarcado requer que tenhamos controle sobre o tempo, bem como controle sobre o ambiente no qual está inserido o sistema, e também a habilidade de acompanhar a execução do programa a fim de detectar erros – é um processo mais complexo que aquele realizado em *desktop*.

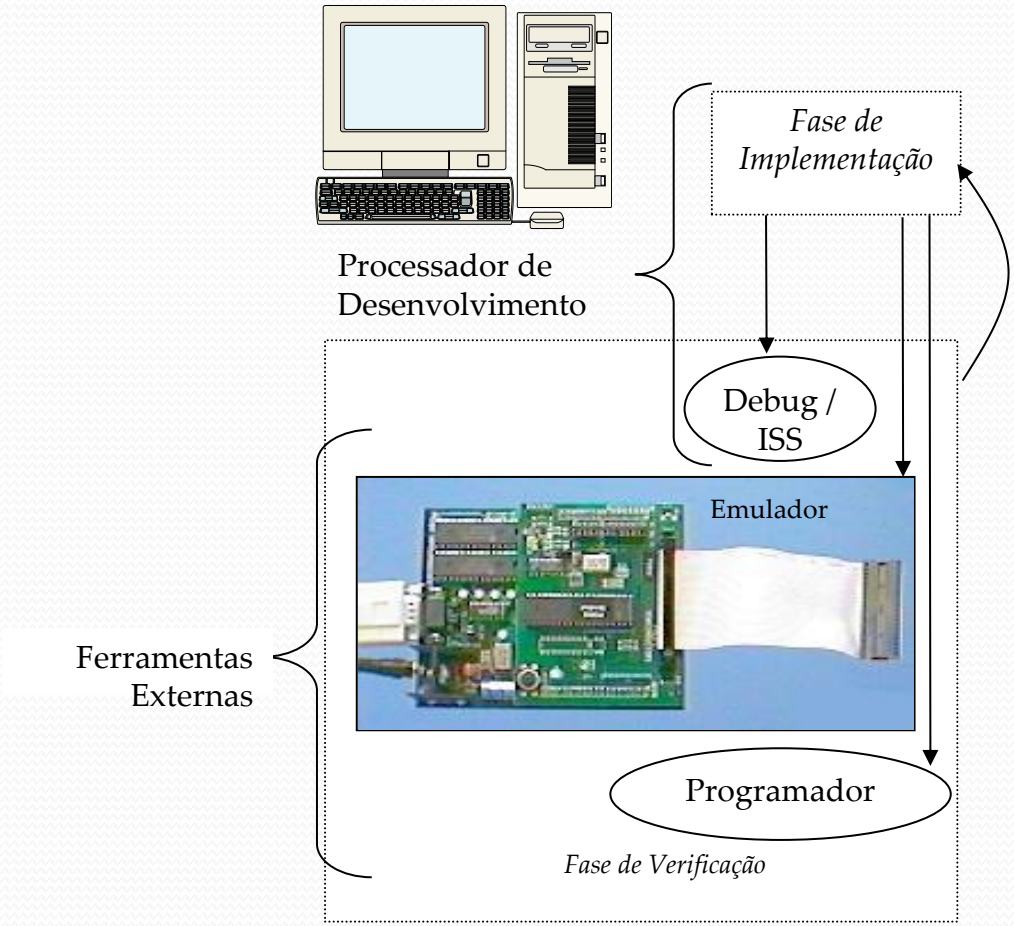
- **ISS (Instruction set simulator):** roda no processador de desenvolvimento, mas executa código projetado para o processador alvo – imita ou simula a função do processador alvo (também chamado de máquina virtual).

- **Emulador:** suporta a depuração do programa enquanto ele é executado no processador alvo. Normalmente, consiste de um depurador acoplado a uma placa conectada ao desktop e que contém o processador alvo e um circuito adicional de suporte.

- **Programadores de dispositivo:** Carregam um programa da memória do processador de desenvolvimento para o processador alvo.

Ambiente de Desenvolvimento

- Em um sistema embarcado:



Ambiente de Desenvolvimento

- Três maneiras de testar o sistema embarcado:
  - Depuração usando ISS – **menos realista e impreciso na observação do comportamento** / **abordagem mais rápida e simples**.
  - Emulação usando um emulador.
  - Teste de campo através do carregamento do programa diretamente na memória do processador alvo – **mais realista** / **abordagem mais lenta**.

Processadores com aplicação específica

- **ASIPs** (*Application-specific instruction-set processors*): buscam um meio termo entre processadores genéricos e dedicados.

  - Dispositivos programáveis (*software*): vantagens em termos de flexibilidade e tempo para o mercado.
  - Desempenho, tamanho e consumo de potência são satisfatórios.
  - Não exigem o alto custo NRE de um processador dedicado.

Microcontroladores

- Podem incluir vários atributos, como:
  - Dispositivos periféricos (temporizadores, conversores A/D e D/A, comunicação serial, etc.) no mesmo circuito integrado que o processador.
  - Memória de dados e de programa no mesmo IC – implementação compacta e de baixa potência.
  - Acesso direto a um número de pinos do IC – facilita o monitoramento de sensores e ajuste/acionamento de atuadores.
  - Instruções especializadas para operações de controle comuns (e.g., manipulação de bits).

Processadores digitais de sinais (DSPs)

- Altamente otimizados para o processamento de largas quantidades de dados.
  - Vários registradores, blocos de memória, multiplicadores e outras unidades aritméticas.
  - Acumulação/Multiplicação em uma única instrução.
  - Realiza operações vetoriais de forma eficiente – e.g., soma de dois vetores.
  - Permitem a execução em paralelo de algumas funções.
  - Operações aritméticas frequentemente utilizadas são implementadas em hardware, reduzindo o tempo de execução.

Como escolher um processador?

- **Critérios:**
  - Técnicos: velocidade, potência consumida, tamanho, custo.
  - Outros: ambiente de desenvolvimento, familiaridade, autorização para uso, etc.

- **Velocidade**
  - Aspecto relativamente difícil de ser medido e comparado.
  - Tentativas:
    - Velocidade do relógio – mas o número de instruções por ciclo de relógio pode ser diferente.
    - Instruções por segundo – mas o trabalho realizado (ou a complexidade) das instruções pode ser diferente.

Como escolher um processador?

- **Benchmarks:** tentativa de criar um mecanismo para comparação "justa" entre diferentes processadores.

  - Dhrystone: *Synthetic Benchmark* – conjunto de programas sintéticos de avaliação, desenvolvido em 1984 – medida em Dhrystones/segundo.

  - MIPS: 1 MIPS = 1757 Dhrystones/segundo (baseado no VAX 11/780 de Digital).
    - Amplamente utilizado hoje em dia.
    - Então, 750 MIPS = 750*1757 = 1.317.750 Dhrystones/segundos.

Como escolher um processador?

Processor	Clock speed	Periph.	Bus Width	MIPS	Power	Trans.	Price
General Purpose Processors							
Intel PIII	1GHz	2x16 K L1, 256K L2, MMX	32	~900	97W	~7M	$900
IBM PowerPC 750X	550 MHz	2x32 K L1, 256K L2	32/64	~1300	5W	~7M	$900
MIPS R5000	250 MHz	2x32 K 2 way set assoc.	32/64	NA	NA	3.6M	NA
StrongARM SA-110	233 MHz	None	32	268	1W	2.1M	NA
Microcontroller							
Intel 8051	12 MHz	4K ROM, 128 RAM, 32 I/O, Timer, UART	8	~1	~0.2W	~10K	$7
Motorola 68HC811	3 MHz	4K ROM, 192 RAM, 32 I/O, Timer, WDT, SPI	8	~.5	~0.1W	~10K	$5
Digital Signal Processors							
TI C5416	160 MHz	128K, SRAM, 3 T1 Ports, DMA, 13 ADC, 9 DAC	16/32	~600	NA	NA	$34
Lucent DSP32C	80 MHz	16K Inst., 2K Data, Serial Ports, DMA	32	40	NA	NA	$75

Projeto do processador genérico

- Processador genérico = processador dedicado cujo propósito é processar instruções armazenadas em uma memória de programa.

- É possível utilizar a técnica de projeto de processadores dedicados, vista no tópico anterior, para construir um processador genérico.

Projeto do processador genérico

- Conjunto de instruções

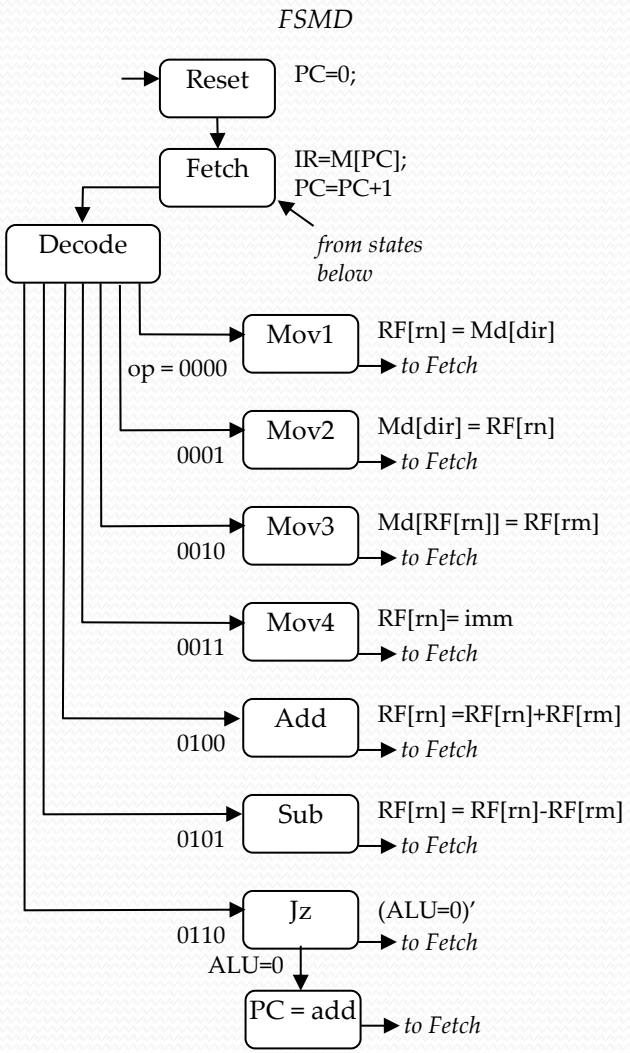
Instrução Assembly	Byte mais significativo	Byte menos significativo	Operação
MOV Rn, direct	0000 | Rn	direct	Rn = Md(direct)
MOV direct, Rn	0001 | Rn	direct	Md(direct) = Rn
MOV @Rn, Rm	0010 | Rn	Rm |	Md(Rn) = Rm
MOV Rn, #immed.	0011 | Rn	immediate	Rn = immediate
ADD Rn, Rm	0100 | Rn	Rm |	Rn = Rn + Rm
SUB Rn, Rm	0101 | Rn	Rm |	Rn = Rn - Rm
JZ #address	0110 |	address	PC = address (somente se o flag de resultado nulo da ALU estiver ativo)

Projeto do processador genérico

- FSMD

  - PC – 8 bits;
  - IR – 16 bits;
  - Memória de programa Mp: 256 x 16;
  - Memória de dados Md: 256 x 8;
  - Arquivo de regist. (RF): 16 x 8.

  - op: IR[15…12]
  - rn (regist. destino): IR[11…8]
  - rm (regist. origem): IR[7…4]
  - dir: IR[7…0]
  - imm : IR[7…0]
  - add : IR[7…0]



Projeto do processador genérico

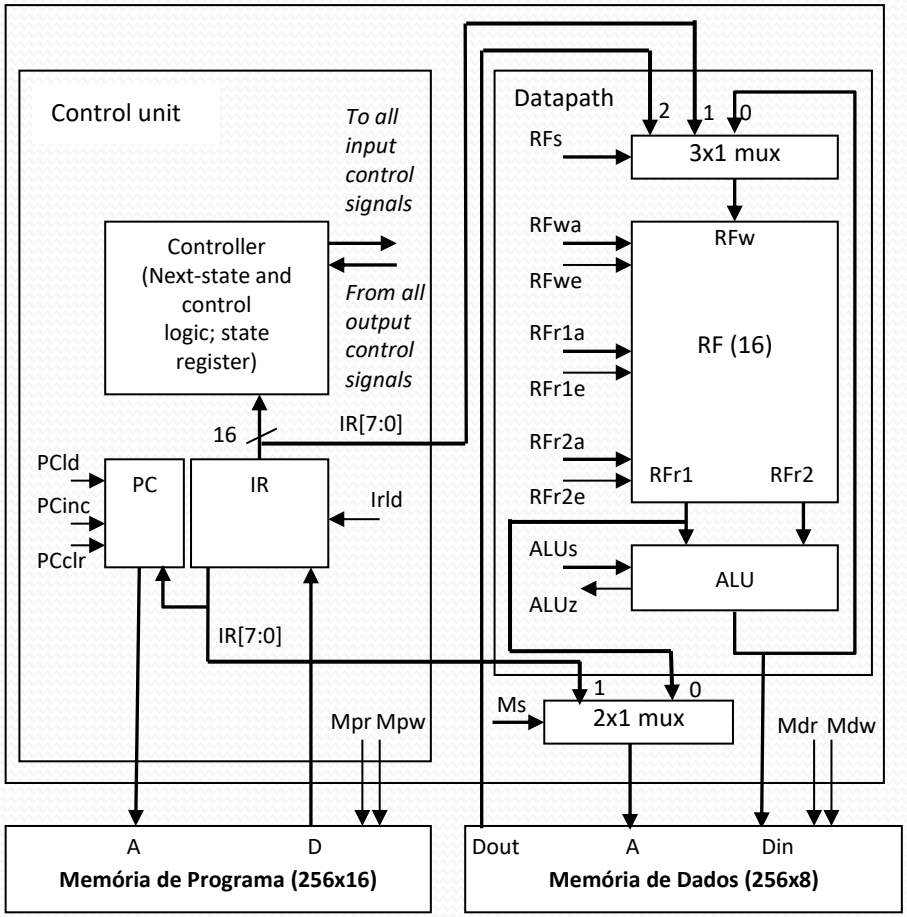
- *Datapath*

- Para cada varíavel declarada, crio um dispositivo de armazenamento (regist. PC e IR, memória M e arquivo de registradores RF).

- Unidades funcionais para executar as operações – uso de uma única ALU.

- Adiciono conexões entre as portas dos componentes como exigido pela FSMD, acrescentando multiplexadores quando há mais de uma conexão sendo colocada em alguma entrada.

- Crio identificadores únicos para todos os sinais de controle.

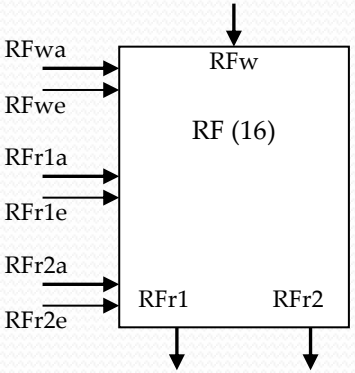


Projeto do processador genérico

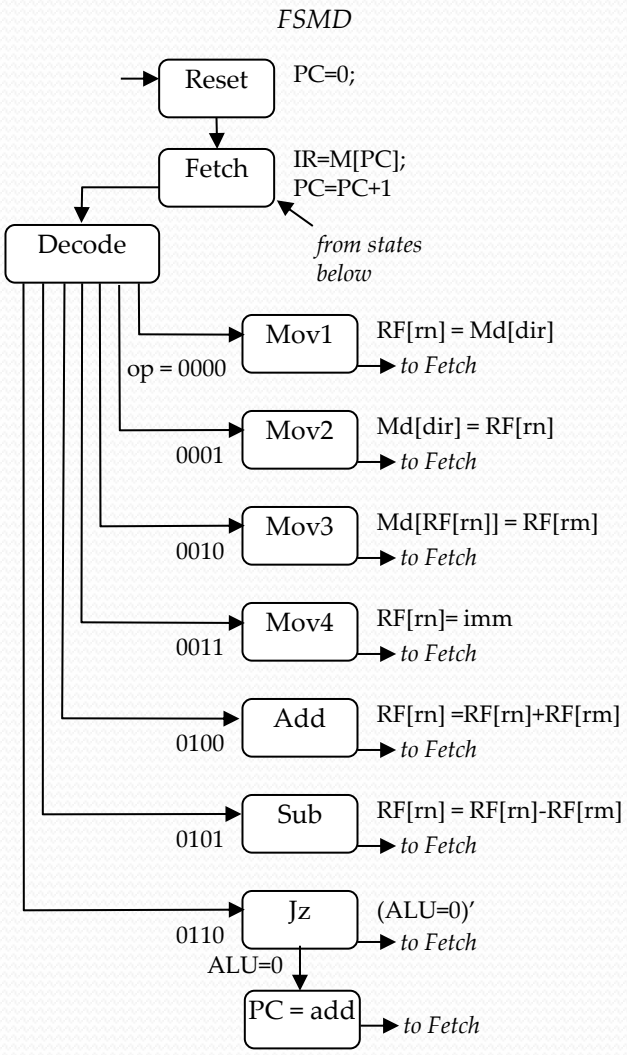
- *Arquivo de registradores RF*

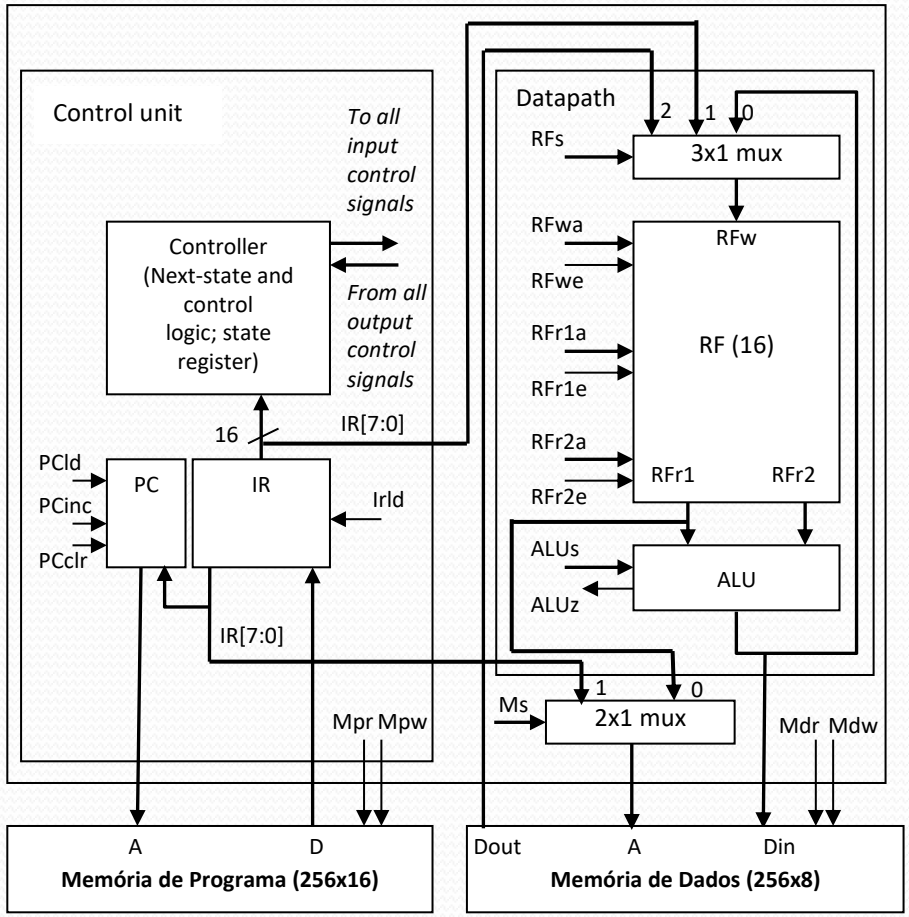
- **Sinais de controle de escrita:**
- RFw (8 bits): valor a ser armazenado em um dos registradores do banco.
- RFwa (4 bits): endereço para seleção do registrador que receberá o valor contido na entrada RFw.
- RFwe (1 bit): sinal que habilita a escrita em um dos registradores do banco.

- **Sinais de controle de leitura:**
- RFr1 (8 bits): conteúdo armazenado no registrador selecionado pelo endereço RFr1a.
- RFr1a (4 bits): endereço para seleção do registrador cujo conteúdo será exibido na saída RFr1.
- RFr1e (1 bit) sinal que habilita a leitura do conteúdo de um registrador na saída RFr1.
- RFr2 (8 bits): conteúdo armazenado no registrador selecionado pelo endereço RFr2a.
- RFr2a (4 bits): endereço para seleção do registrador cujo conteúdo será exibido na saída RFr2.
- RFr2e (1 bit) sinal que habilita a leitura do conteúdo de um registrador na saída RFr2.

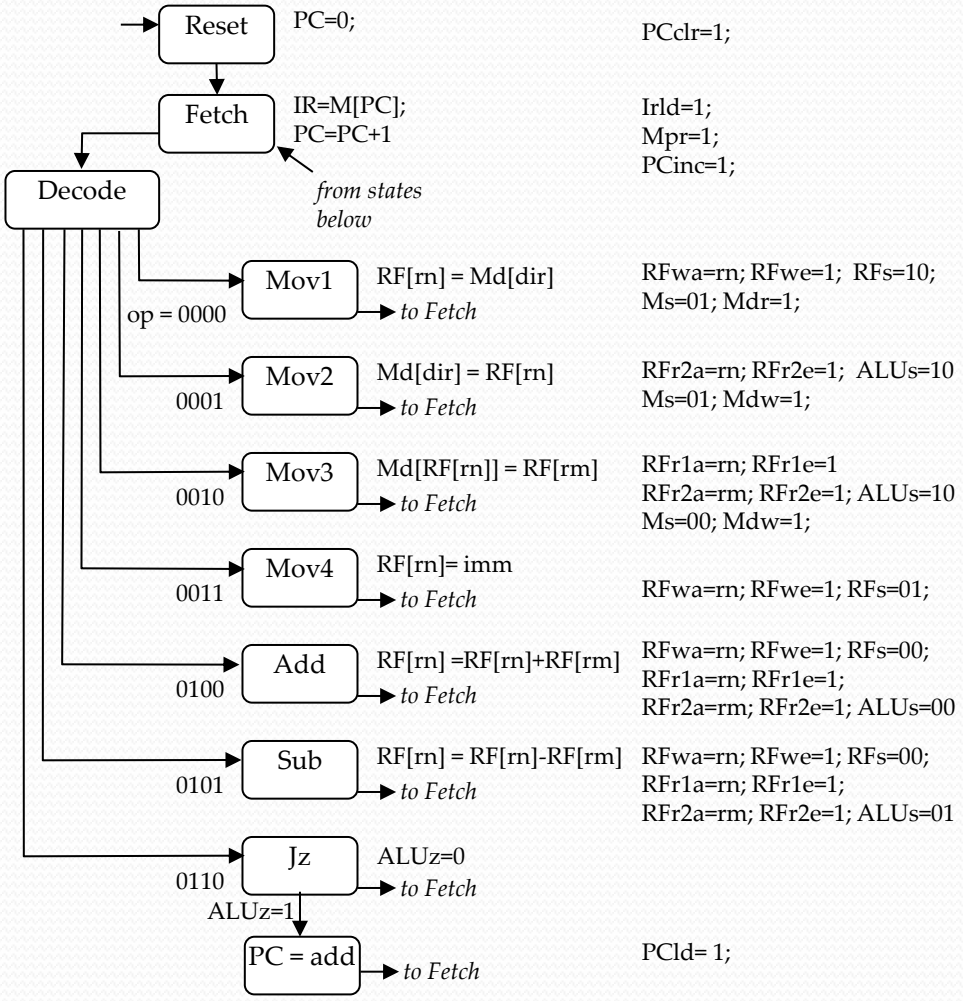


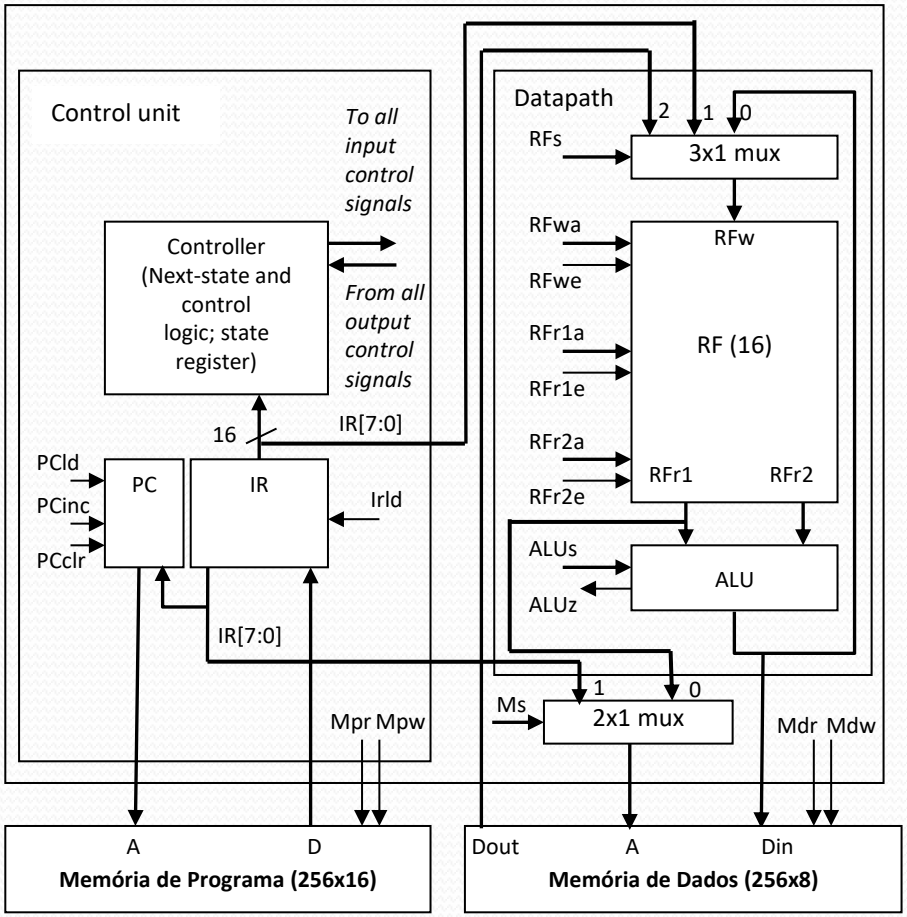
Projeto do processador genérico





Projeto do processador genérico





ALUs = 00: RFr1 + RFr2; ALUs = 01: RFr1 – RFr2; ALUs = 10: copia RFr2 na saída da ALU

Processador genérico x dedicado

- A diferença é que o processador dedicado põe o "programa" dentro de sua lógica de controle, enquanto um processador genérico o mantém em uma memória externa.

- Uma segunda diferença é que o *datapath* de um processador genérico é projetado sem o conhecimento de qual programa será colocado na memória, enquanto tal conjunto de comandos (programa) é conhecido no caso de um processador dedicado.

Sumário

- Processador de propósito geral
  - Bom desempenho, baixo NRE e flexível
- Estrutura: Controlador, *Datapath* e Memória.
- Pipeline: paralelização da execução das instruções.
- Programação é feita principalmente em linguagem estruturada mas em alguns casos pode ser necessário recorrer à programação assembly.
- Muitas ferramentas de desenvolvimento, incluindo simuladores (ISSs) e emuladores.
- ASIPs: microcontroladores, DSPs e outros.
- O projeto de um processador genérico é conceitualmente equivalente ao projeto de um processador dedicado.