Processamento e Linguagem de Máquina

O processador

Programação Assembler (Linguagem de Máquina)

O PROCESSADOR

Literatura

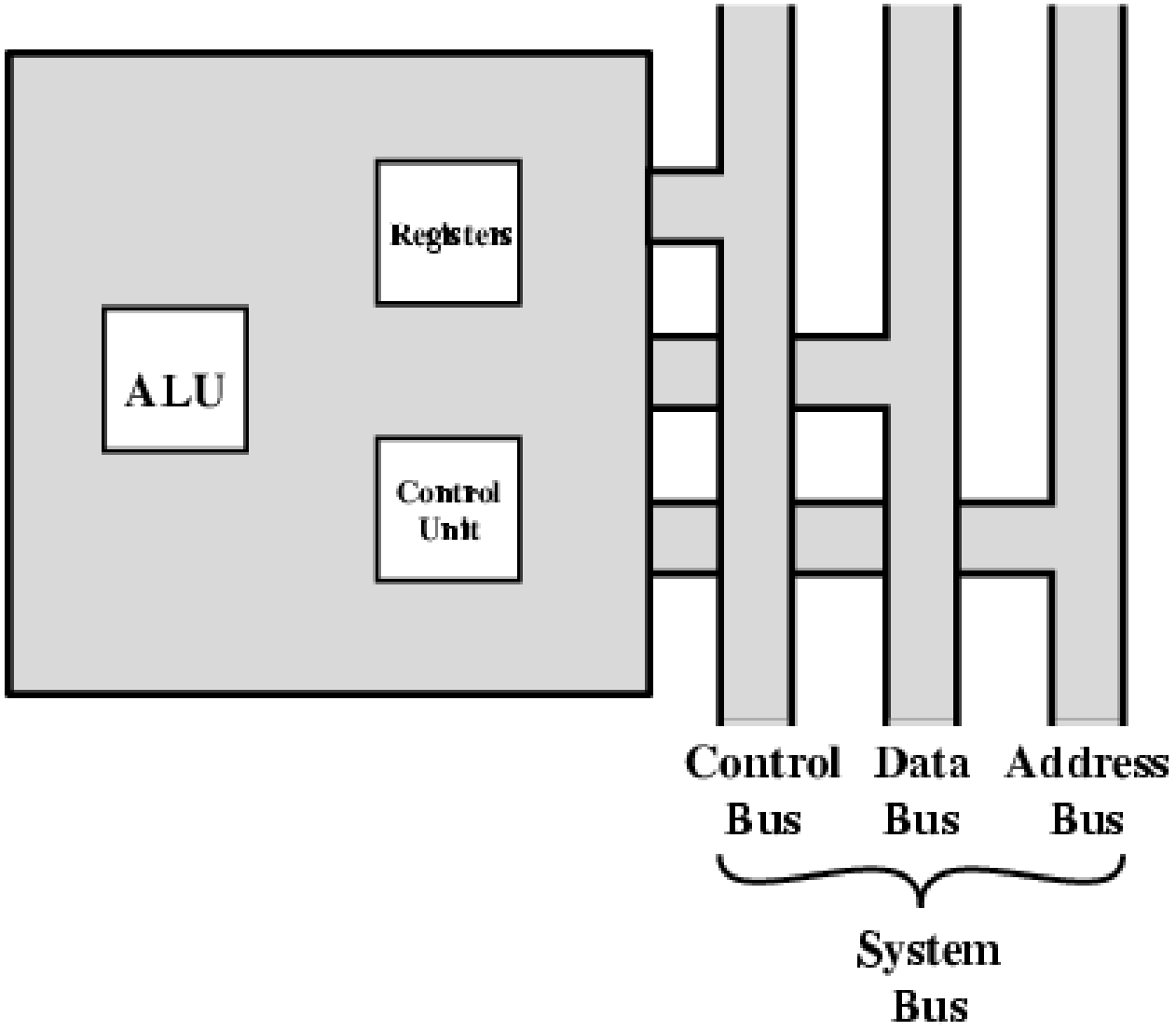
- Tanenbaum e Stallings
  - Abordagens similares mas divergentes
  - Foco maior em funcionamento ou arquitetura
  - Literaturas complementares

- Arquiteturas recentes (online)
  - SOC – System on a chip
  - Hewlett Packard: The Machine

Componentes

- A Unidade de Controle e a Unidade Lógica e Aritmética constituem a Unidade Central de Processamento.
- Dados e instruções precisam entrar no sistema, e resultados saem dele.
  - Entrada/saída.
- É necessário um armazenamento temporário de código e resultados.
  - Memória principal/cache.

CPU e BUS



*Stallings

Registradores

- Espaço de trabalho (entrada/saída) no proc.
- Quantidade (8-32/proc.), capacidade e função dependem do modelo da arquitetura
- Nível mais alto de memória

→ decisão muito importante na arquitetura

Tipos de Registradores

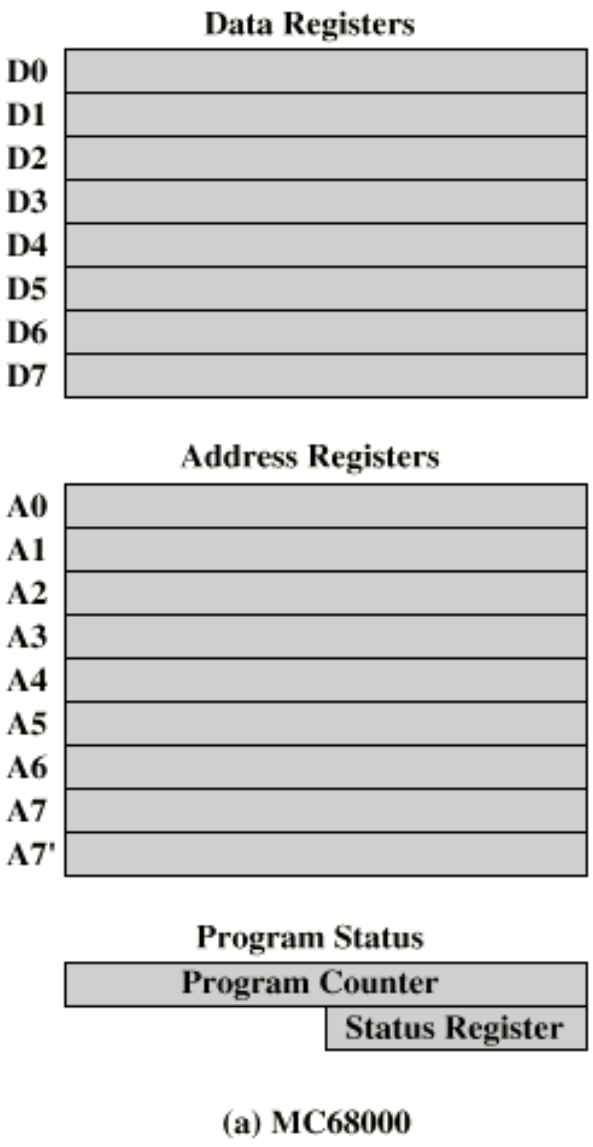
- De propósito geral (podem ser combinados)
- De dados
- De endereços
- De posição de memória de programa
- Indicadores de condição (status)

Registrador de status

Comjunto de bits indicando

- Condition Codes
- Sign of last result
- Zero
- Carry
- Equal
- Overflow
- Interrupt enable/disable
- Supervisor
  - Intel ring zero
  - Kernel mode
  - Allows privileged instructions to execute
  - Used by operating system
  - Not available to user programs

Exemplo de arq. de Registradores



*Stallings

Pentium 4 Registers

(a) Integer Unit

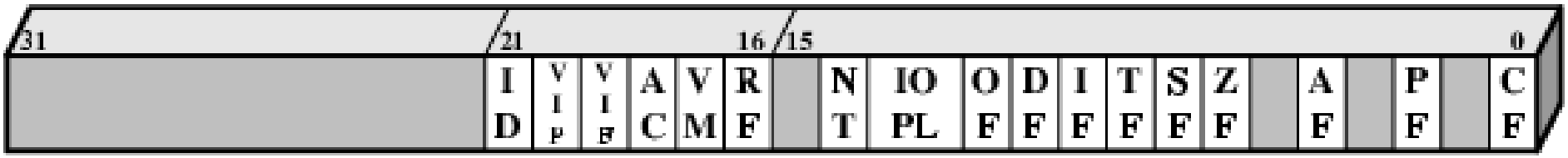
Type	Number	Length (bits)	Purpose
General	8	32	General-purpose user registers
Segment	6	16	Contain segment selectors
Flags	1	32	Status and control bits
Instruction Pointer	1	32	Instruction pointer

(b) Floating-Point Unit

Type	Number	Length (bits)	Purpose
Numeric	8	80	Hold floating-point numbers
Control	1	16	Control bits
Status	1	16	Status bits
Tag Word	1	16	Specifies contents of numeric registers
Instruction Pointer	1	48	Points to instruction interrupted by exception
Data Pointer	1	48	Points to operand interrupted by exception

*Stallings

EFLAGS Register

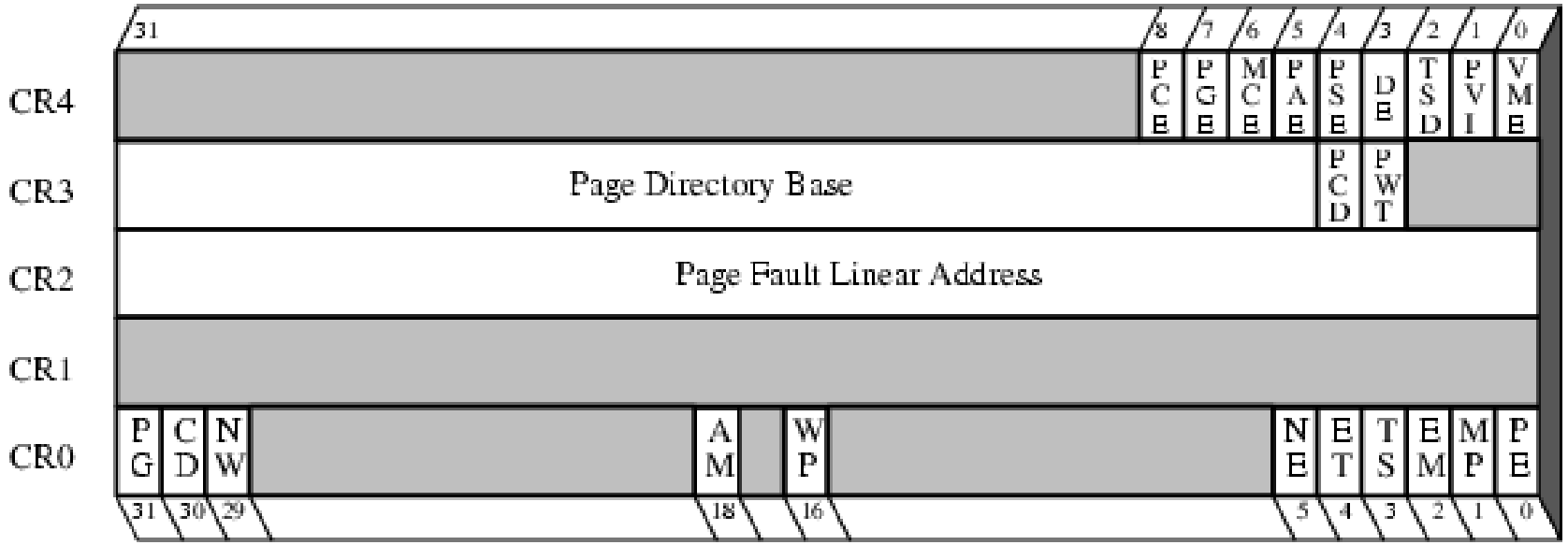


ID = Identification flag
VIP = Virtual interrupt pending
VIF = Virtual interrupt flag
AC = Alignment check
VM = Virtual 8086 mode
RF = Resume flag
NT = Nested task flag
IOPL = I/O privilege level
OF = Overflow flag

DF = Direction flag
IF = Interrupt enable flag
TF = Trap flag
SF = Sign flag
ZF = Zero flag
AF = Auxiliary carry flag
PF = Parity flag
CF = Carry flag

*Stallings

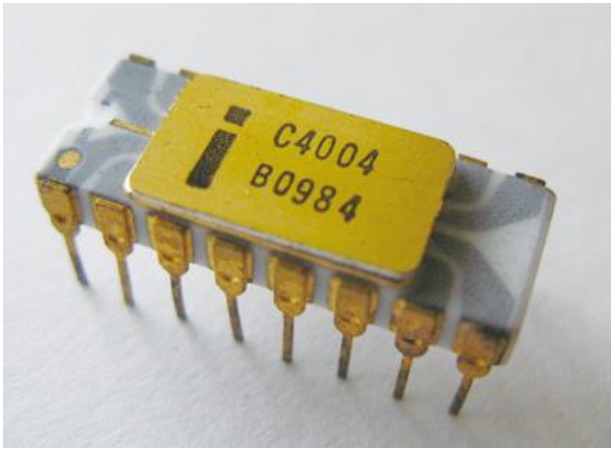
Control Registers

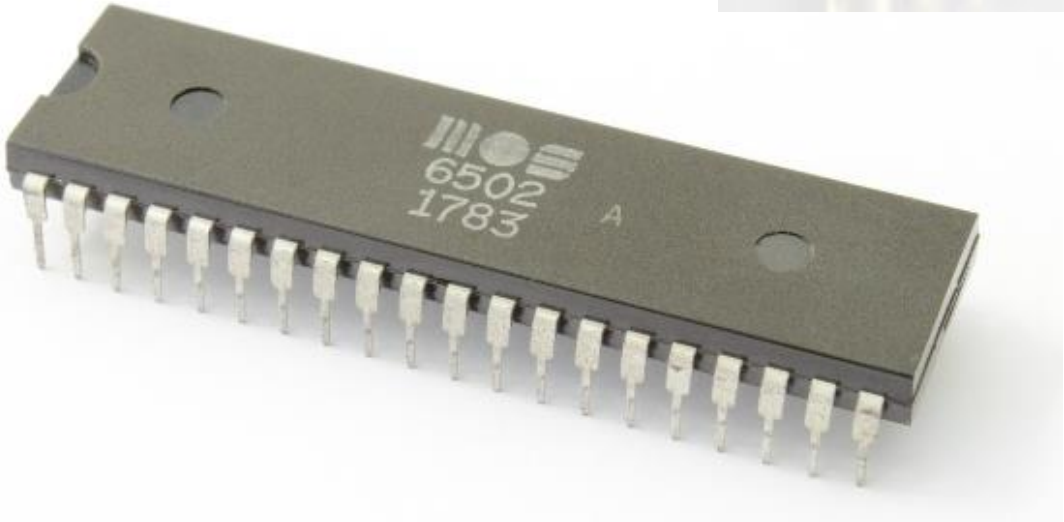


PCE	=	Performance Counter Enable	PG	=	Paging
PGE	=	Page Global Enable	CD	=	Cache Disable
MCE	=	Machine Check Enable	NW	=	Not Write Through
PAE	=	Physical Address Extension	AM	=	Alignment Mask
PSE	=	Page Size Extensions	WP	=	Write Protect
DE	=	Debug Extensions	NE	=	Numeric Error
TSD	=	Time Stamp Disable	ET	=	Extension Type
PVI	=	Protected Mode Virtual Interrupt	TS	=	Task Switched
VME	=	Virtual 8086 Mode Extensions	EM	=	Emulation
PCD	=	Page-level Cache Disable	MP	=	Monitor Coprocessor
PWT	=	Page-level Writes Transparent	PE	=	Protection Enable

*Stallings

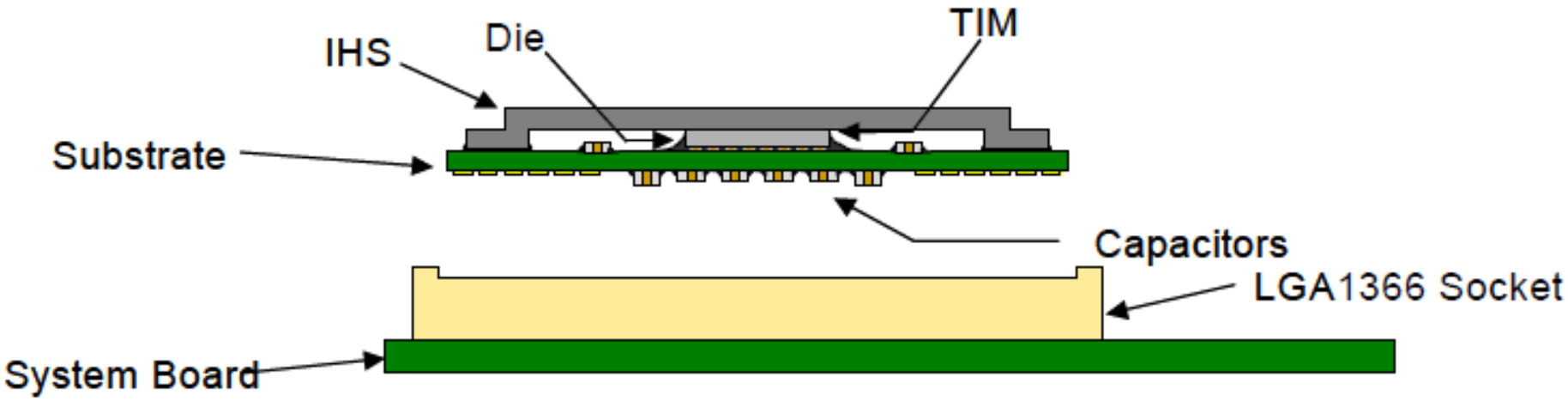
Chips de CPU no passado



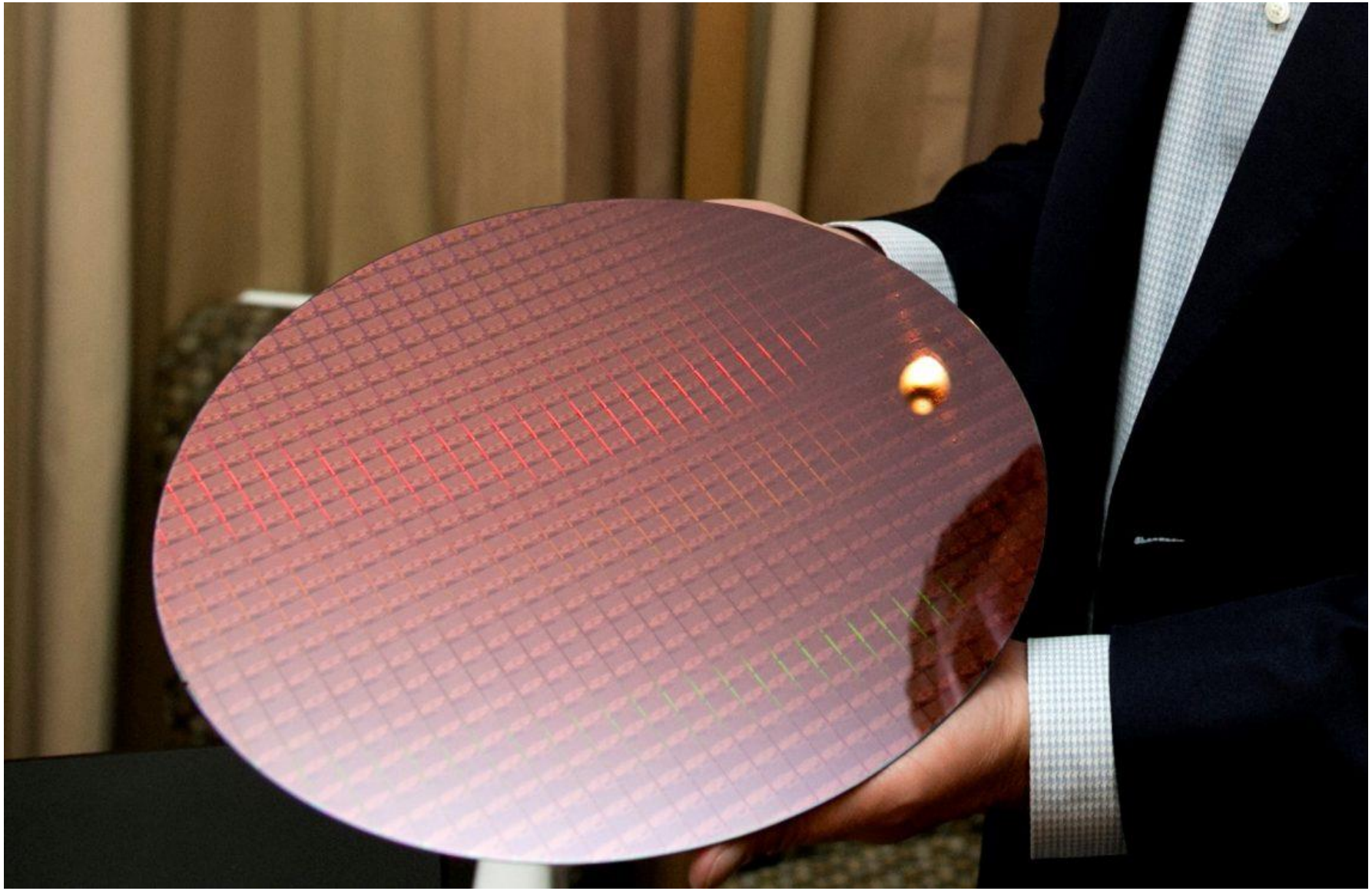


Chips de CPU hoje

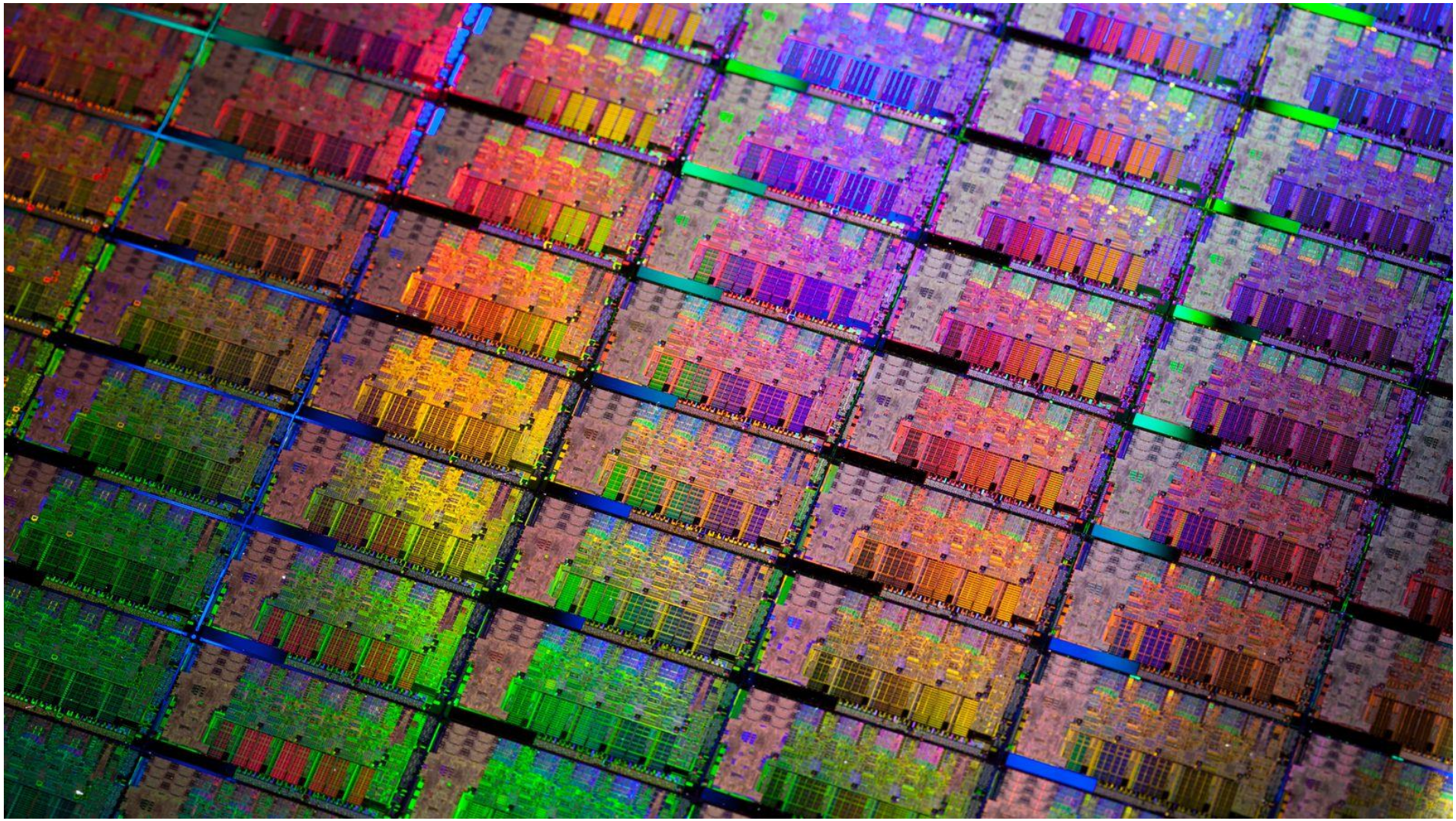
Core CHIP


IHS
Die
TIM
Substrate
Capacitors
LGA1366 Socket
System Board

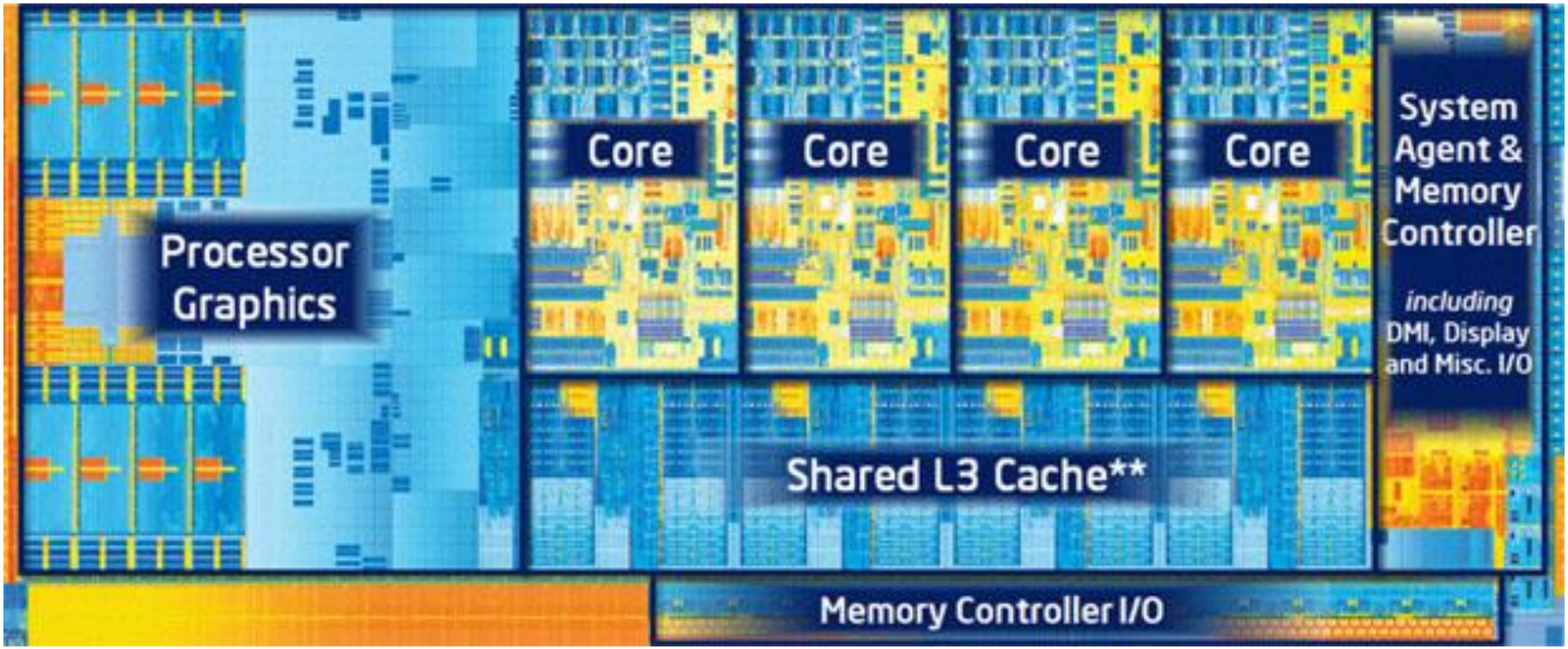
Die



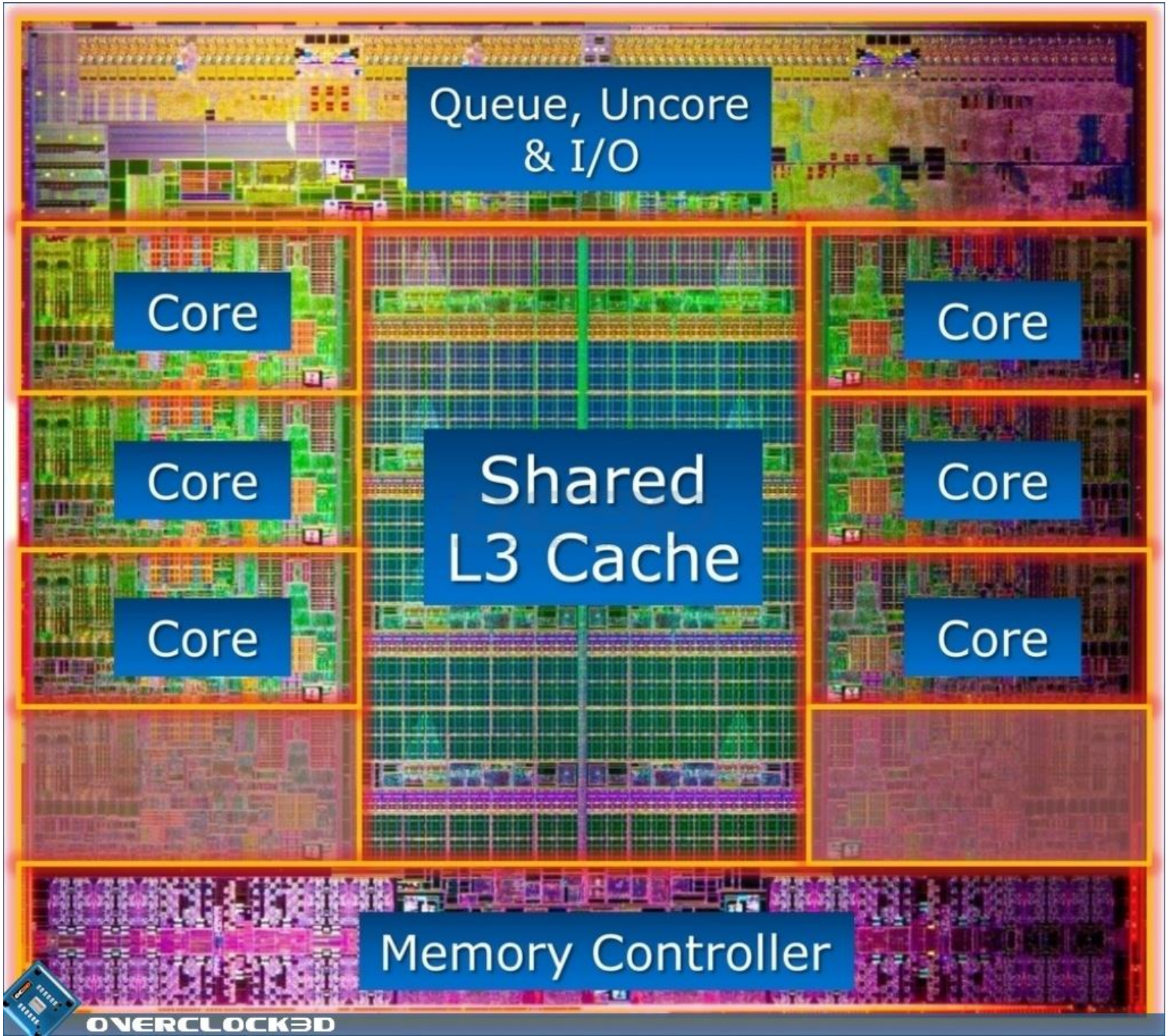
Die



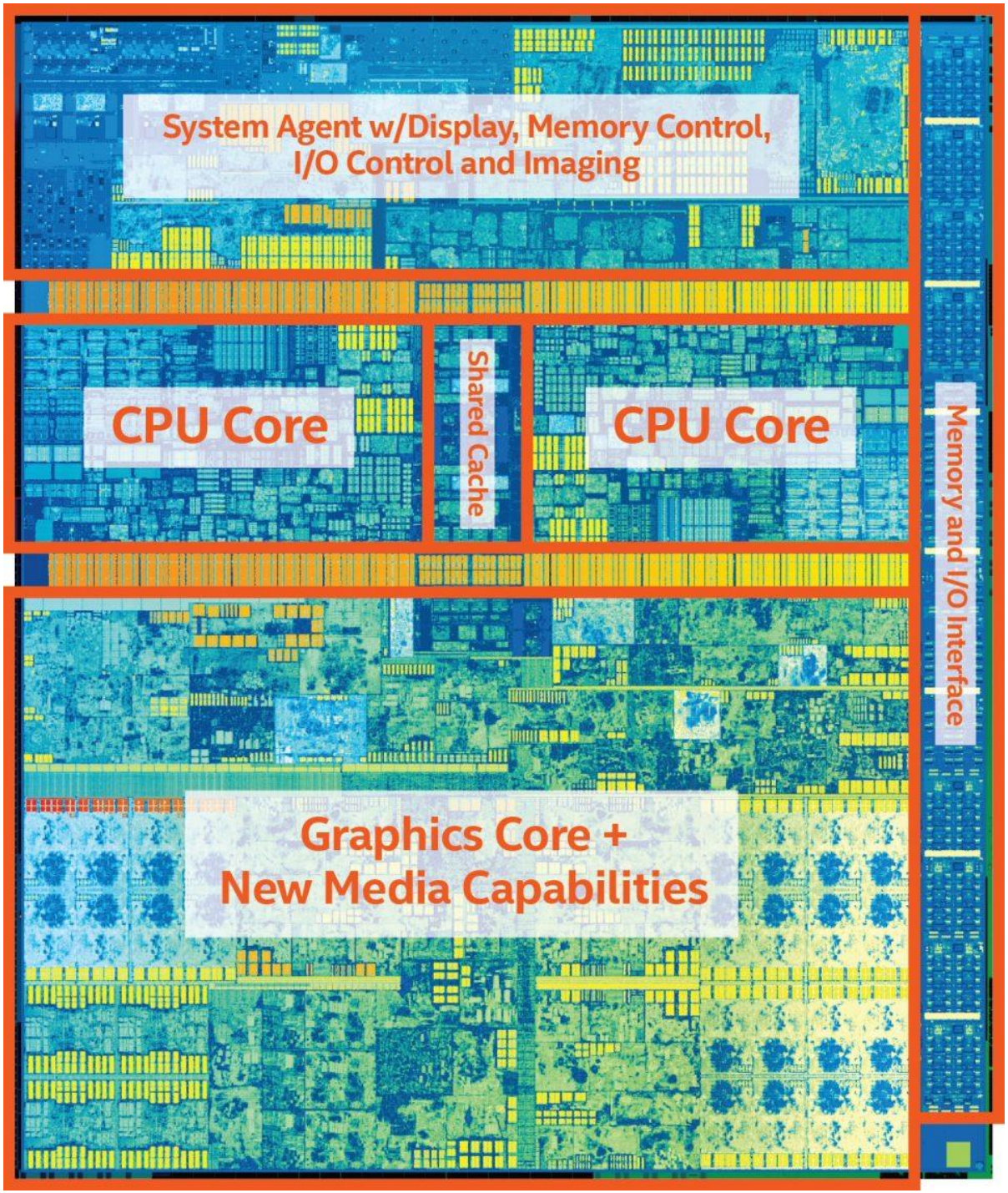
1º g. Core i7 – Ivy Bridge (22nn)



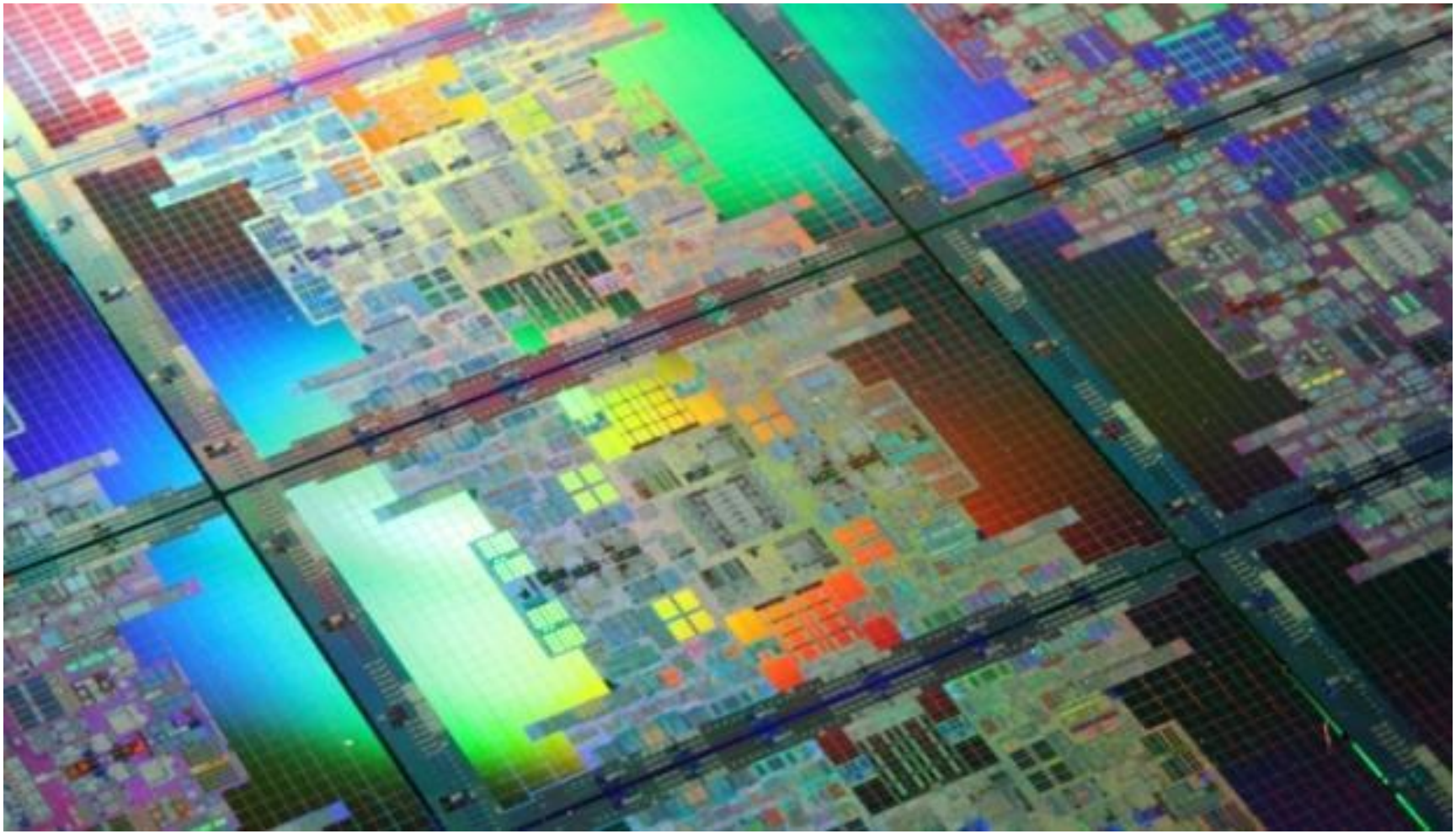
6g


Queue, Uncore
& I/O
Core
Core
Core
Core
Core
Core
Shared
L3 Cache
Memory Controller
OVERCLOCKED

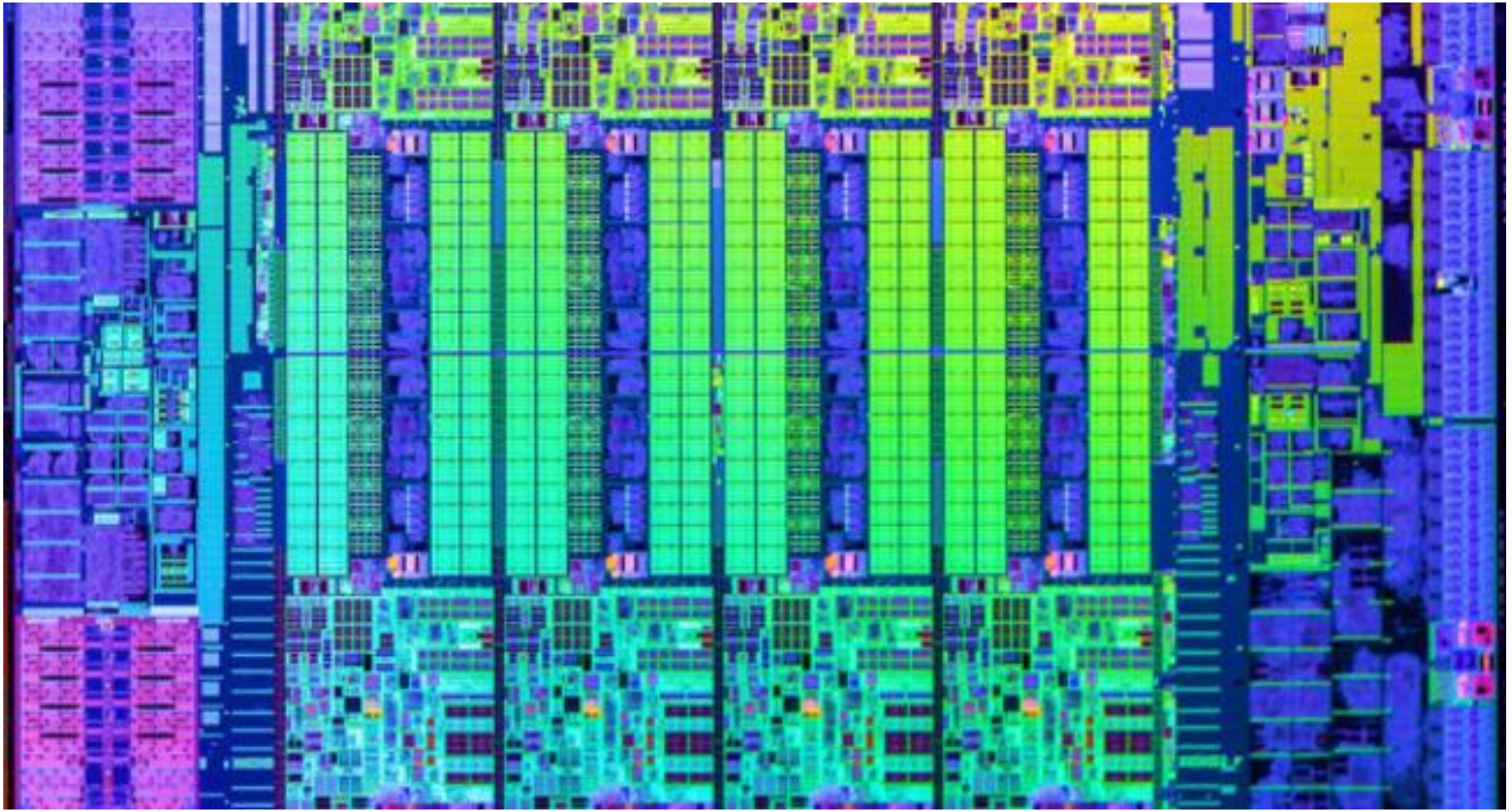
7g


System Agent w/Display, Memory Control, I/O Control and Imaging
CPU Core
Shared Cache
CPU Core
Memory and I/O Interface
Graphics Core + New Media Capabilities

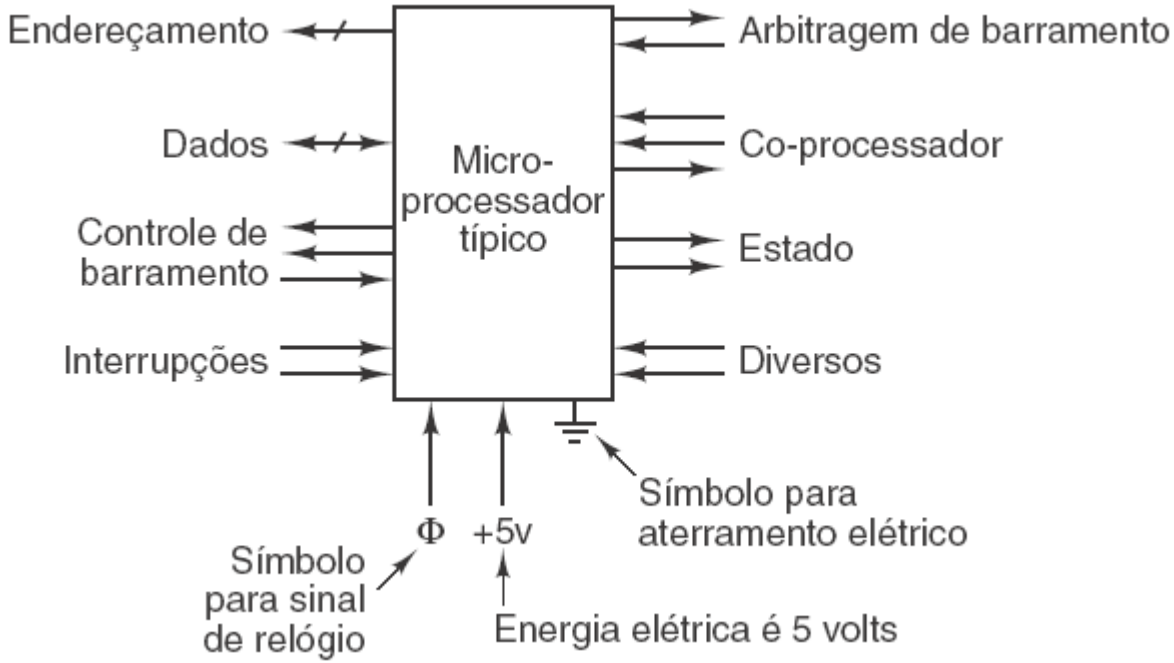
Cannonlake (2017)



Cannonlake (2017)



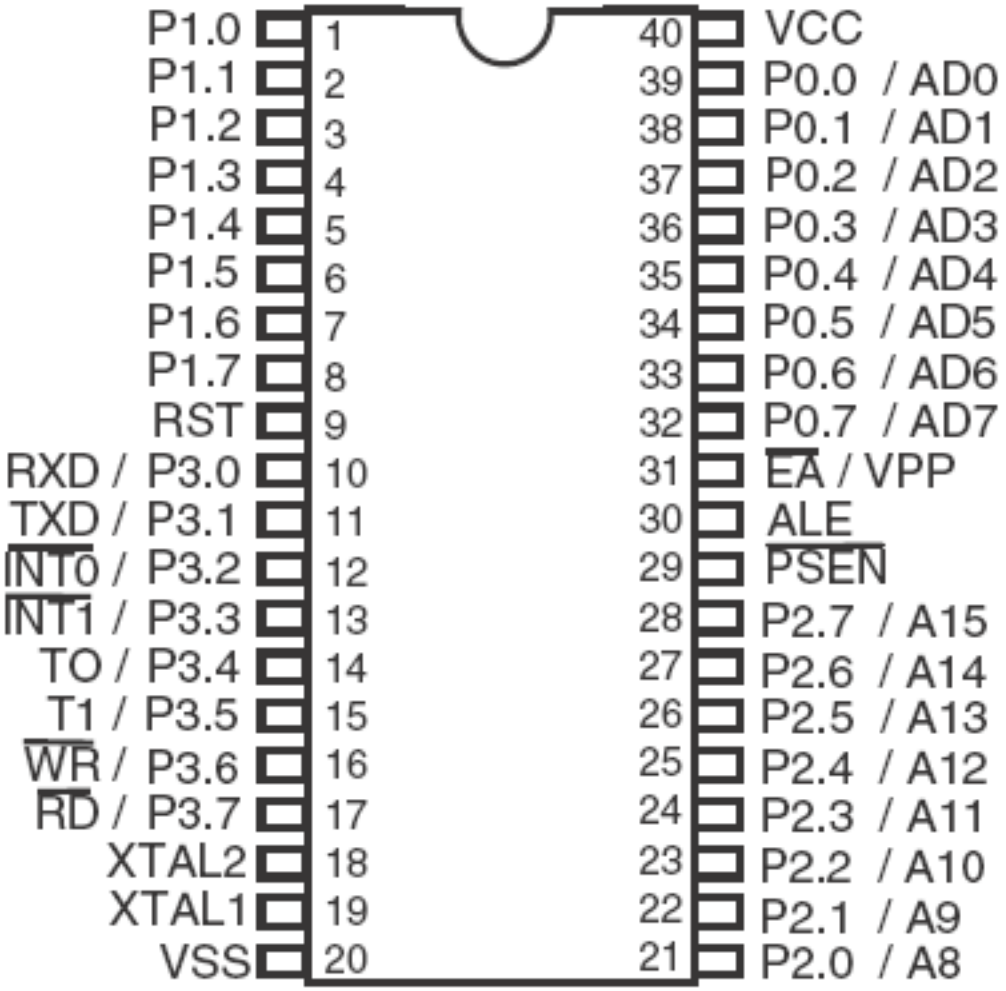
Chips de CPU



Pinagem lógica de uma CPU genérica. As setas indicam sinais de entrada e sinais de saída. Os segmentos de reta diagonal indicam que são utilizados vários pinos. Há um número que indica quantos são os pinos para uma CPU específica.

*Tanenbaum

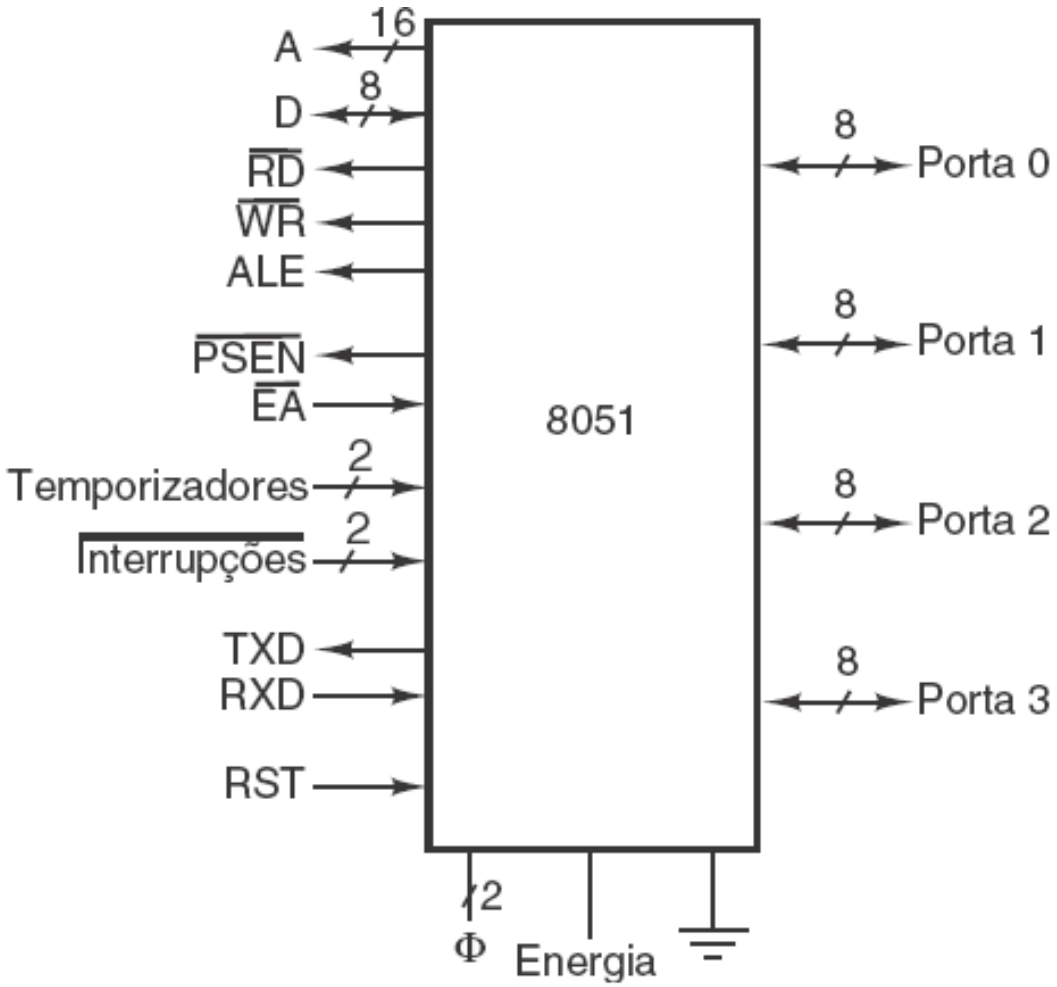
O 8051



Pinagem física do 8051.

*Tanenbaum

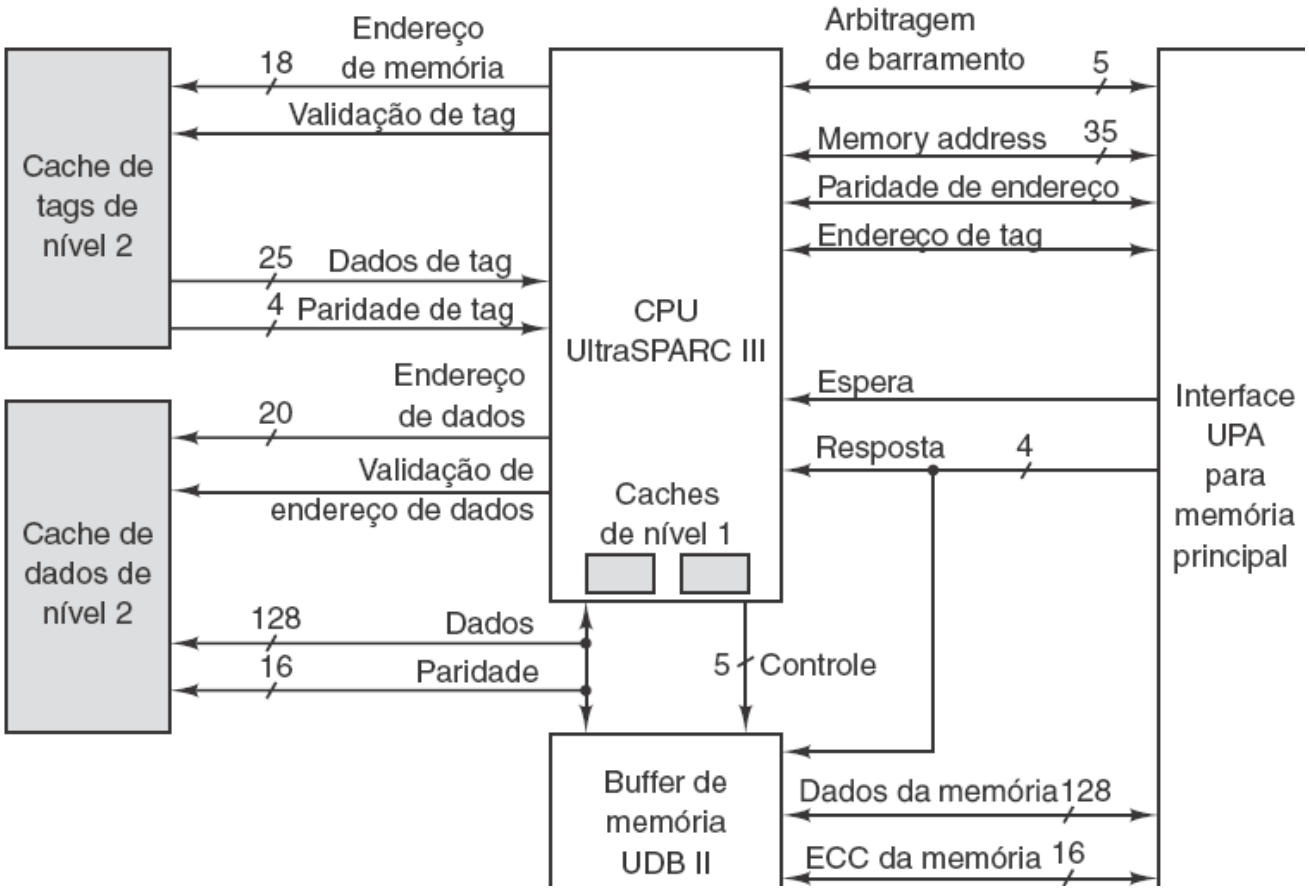
O 8051



Pinagem lógica do 8051.

*Tanenbaum

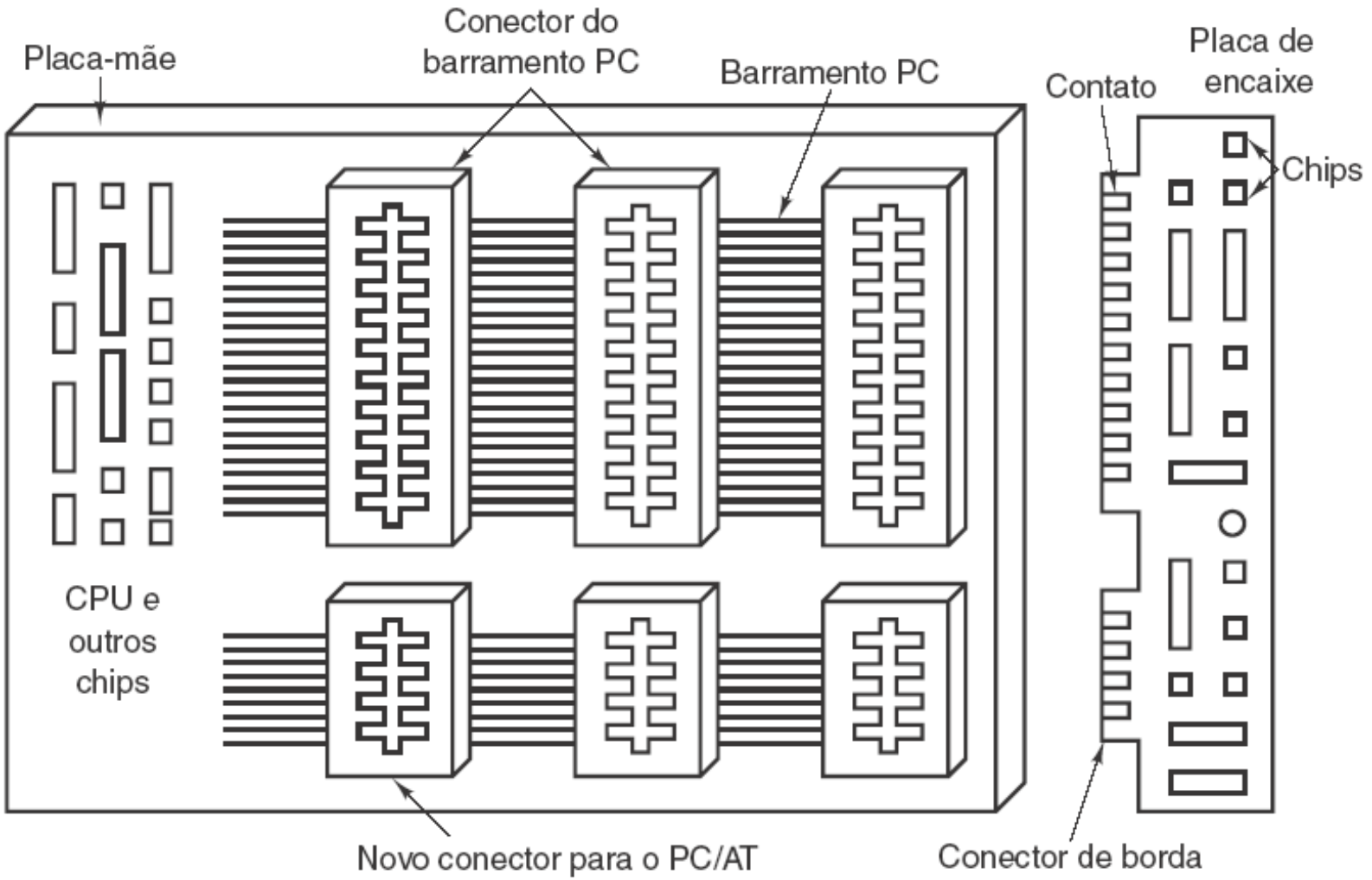
O UltraSPARC III (2)



Principais características do núcleo de um sistema UltraSPARC III.

*Tanenbaum

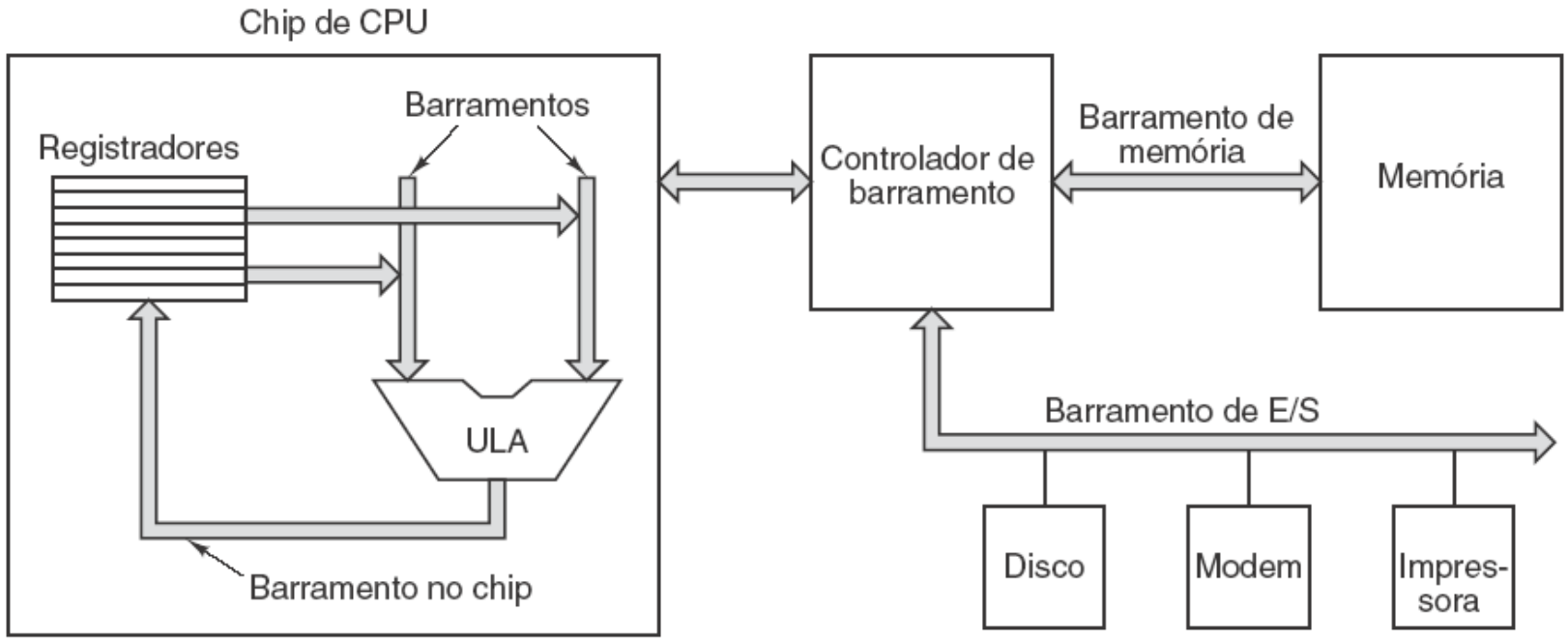
O barramento ISA



O barramento PC/AT tem dois componentes:
a parte do PC original e a parte nova.

*Tanenbaum

Barramentos de computador



Sistema de computador com vários barramentos.

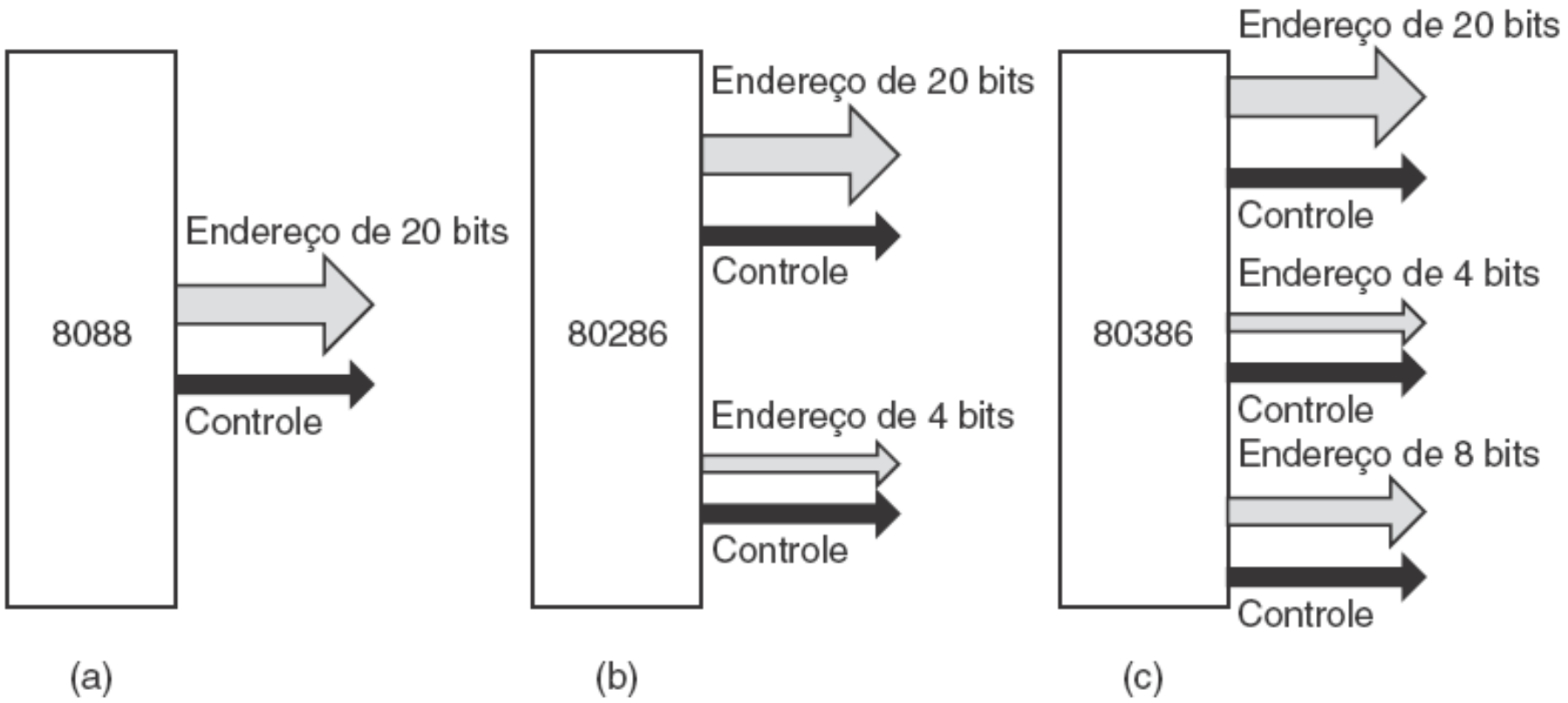
*Tanenbaum

Barramentos de computador

Mestre	Escravo	Exemplo
CPU	Memória	Buscar instruções e dados
CPU	Dispositivo de E/S	Iniciar transferência de dados
CPU	Co-processador	CPU que passa instruções para o co-processador
E/S	Memória	DMA (acesso direto à memória)
Co-processador	CPU	Co-processador que busca operandos na CPU

Exemplos de mestres e escravos de barramentos.

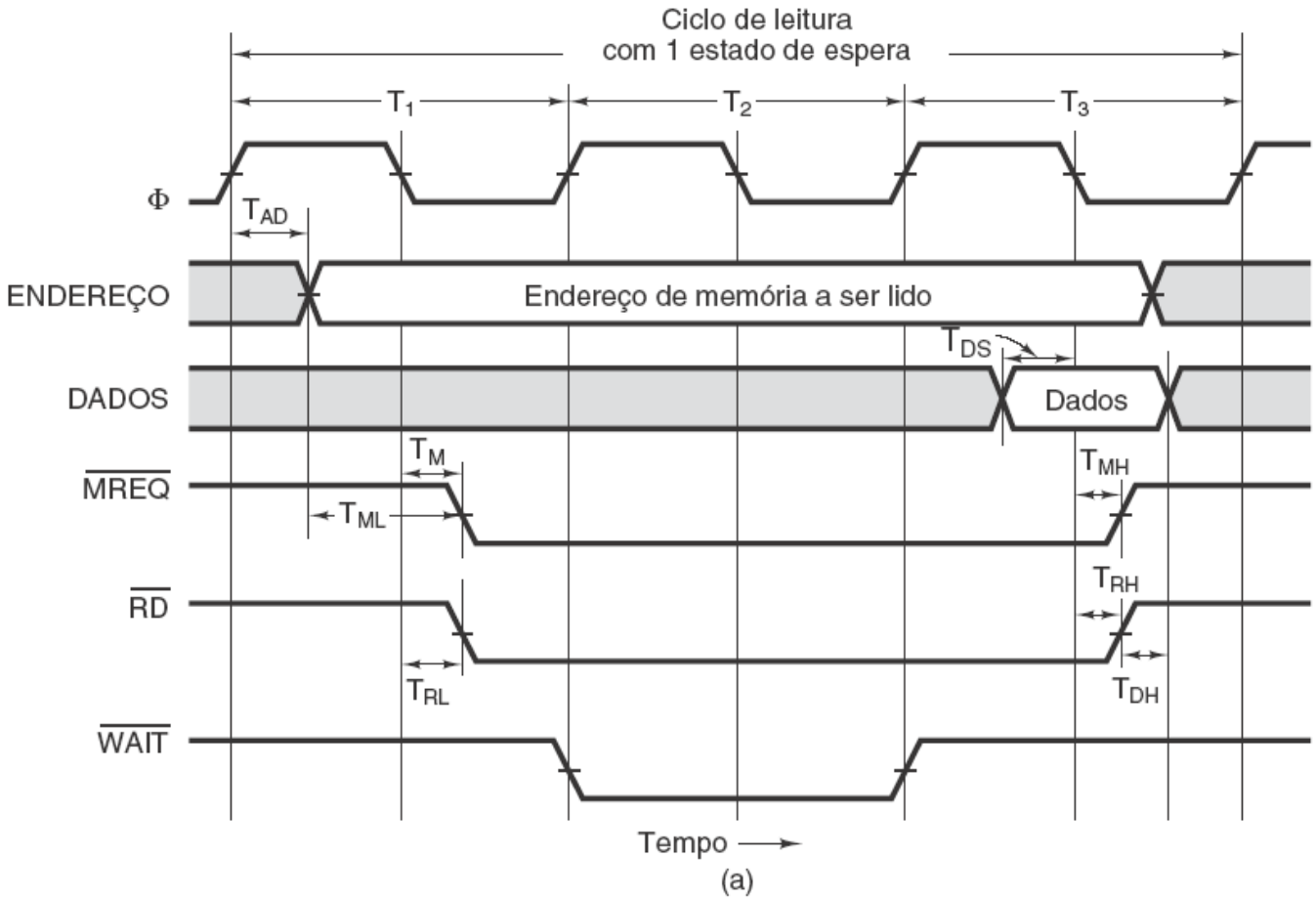
Largura do barramento



Crescimento de um barramento de endereço ao longo do tempo.

*Tanenbaum

Relógio do barramento



Temporização de leitura em um barramento síncrono.

*Tanenbaum

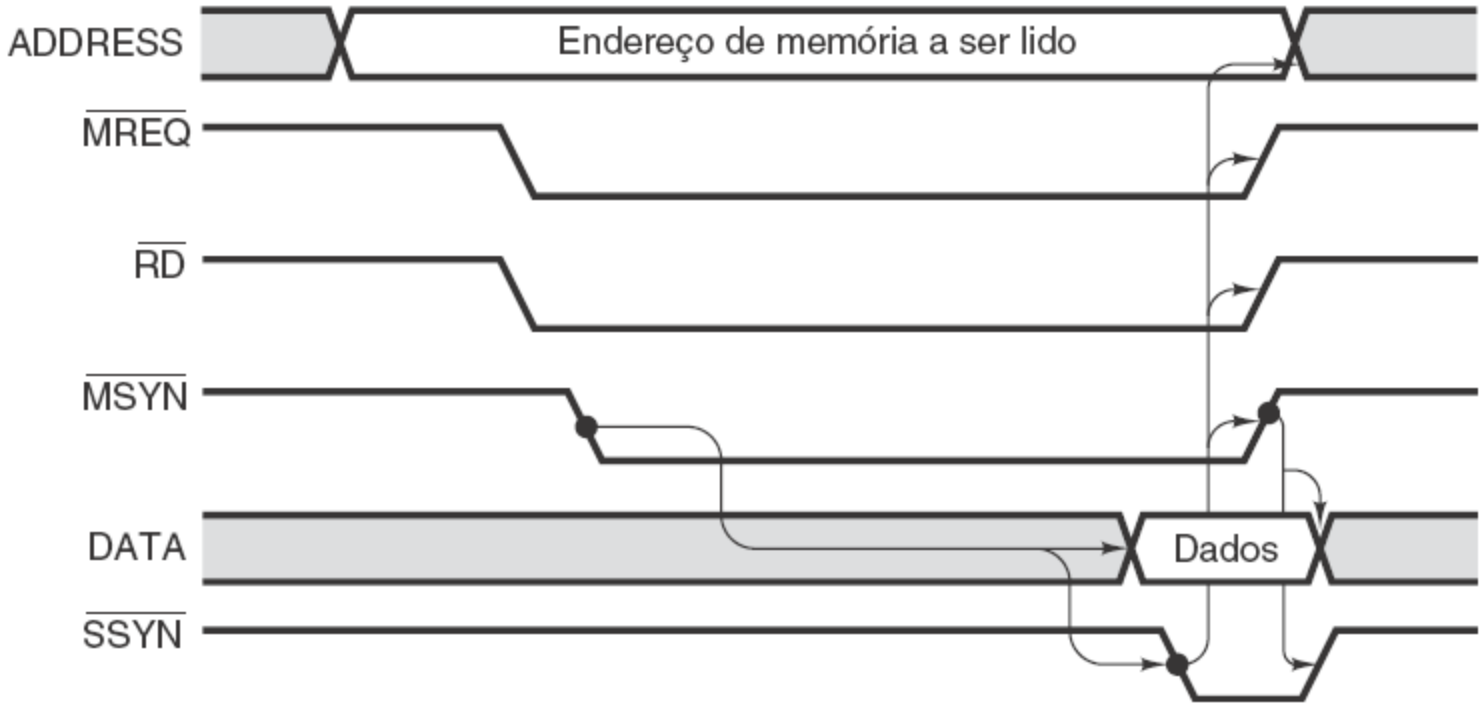
Relógio do barramento

Símbolo	Parâmetro	Mín.	Máx.	Unidade
T_{AD}	Atraso de saída do endereço		4	nsec
T_{ML}	Endereço estável antes de $\overline{MREQ}$	2		nsec
T_M	Atraso de $\overline{MREQ}$ desde a borda descendente de Φ em T1		3	nsec
T_{RL}	Atraso de RD desde a borda descendente de Φ em T1		3	nsec
T_{DS}	Tempo de ajuste dos dados antes da borda descendente de Φ	2		nsec
T_{MH}	Atraso de $\overline{MREQ}$ desde a borda descendente de Φ em T3		3	nsec
T_{RH}	Atraso de $\overline{RD}$ desde a borda descendente de Φ em T3		3	nsec
T_{DH}	Tempo de sustentação dos dados desde a negação de $\overline{RD}$	0		nsec

(b)

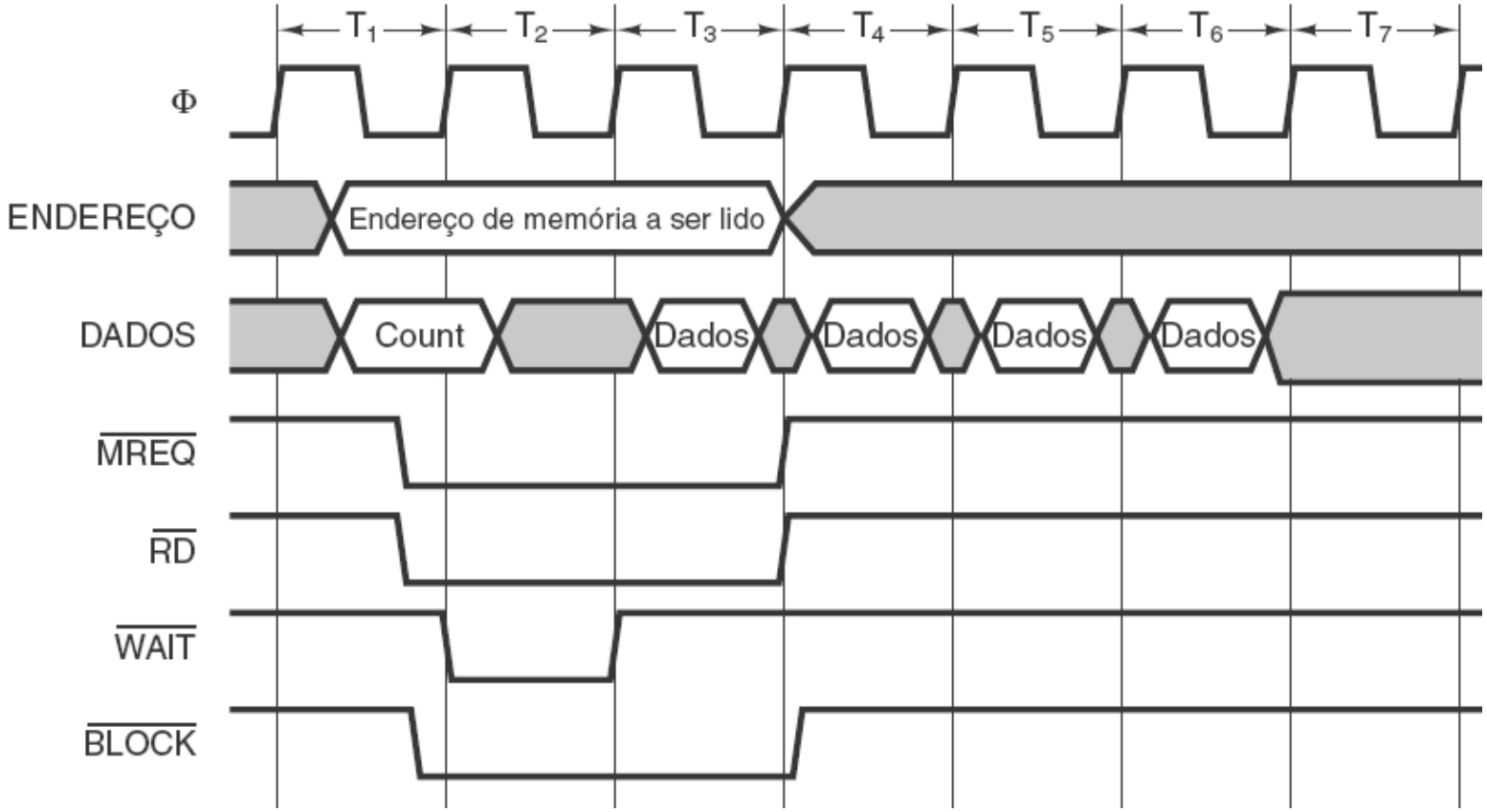
Especificação de alguns tempos críticos.

Barramentos assíncronos



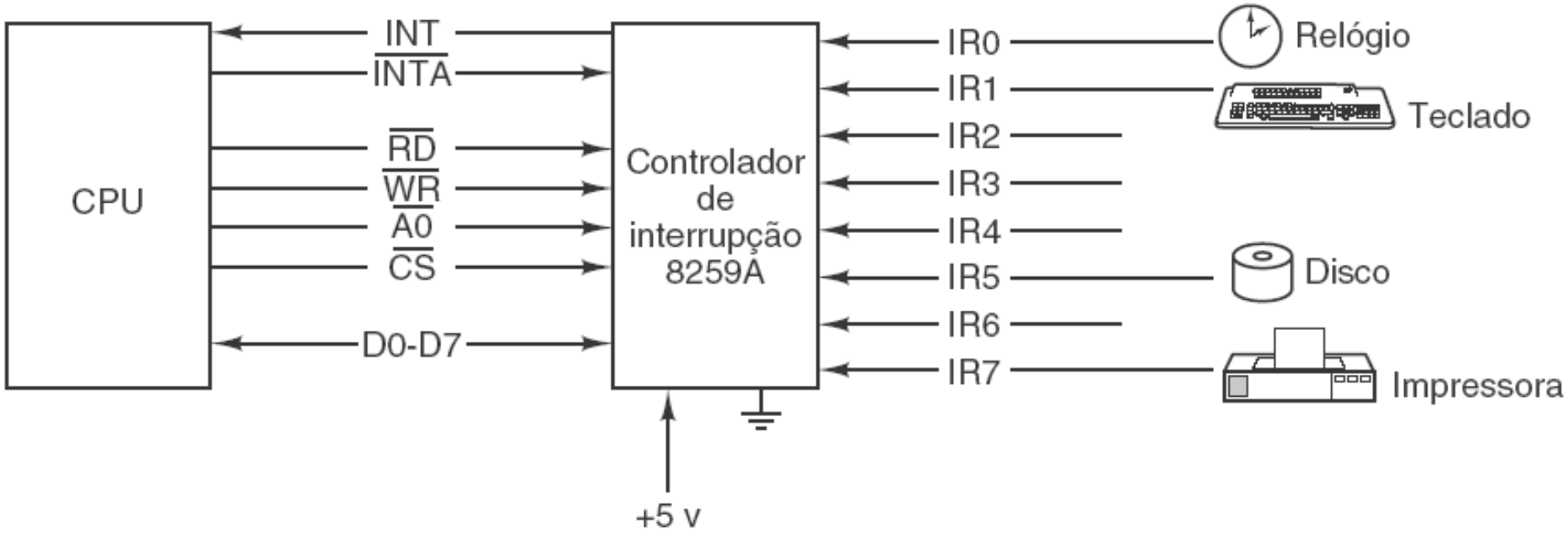
Operação de um barramento assíncrono.

Operações de barramento



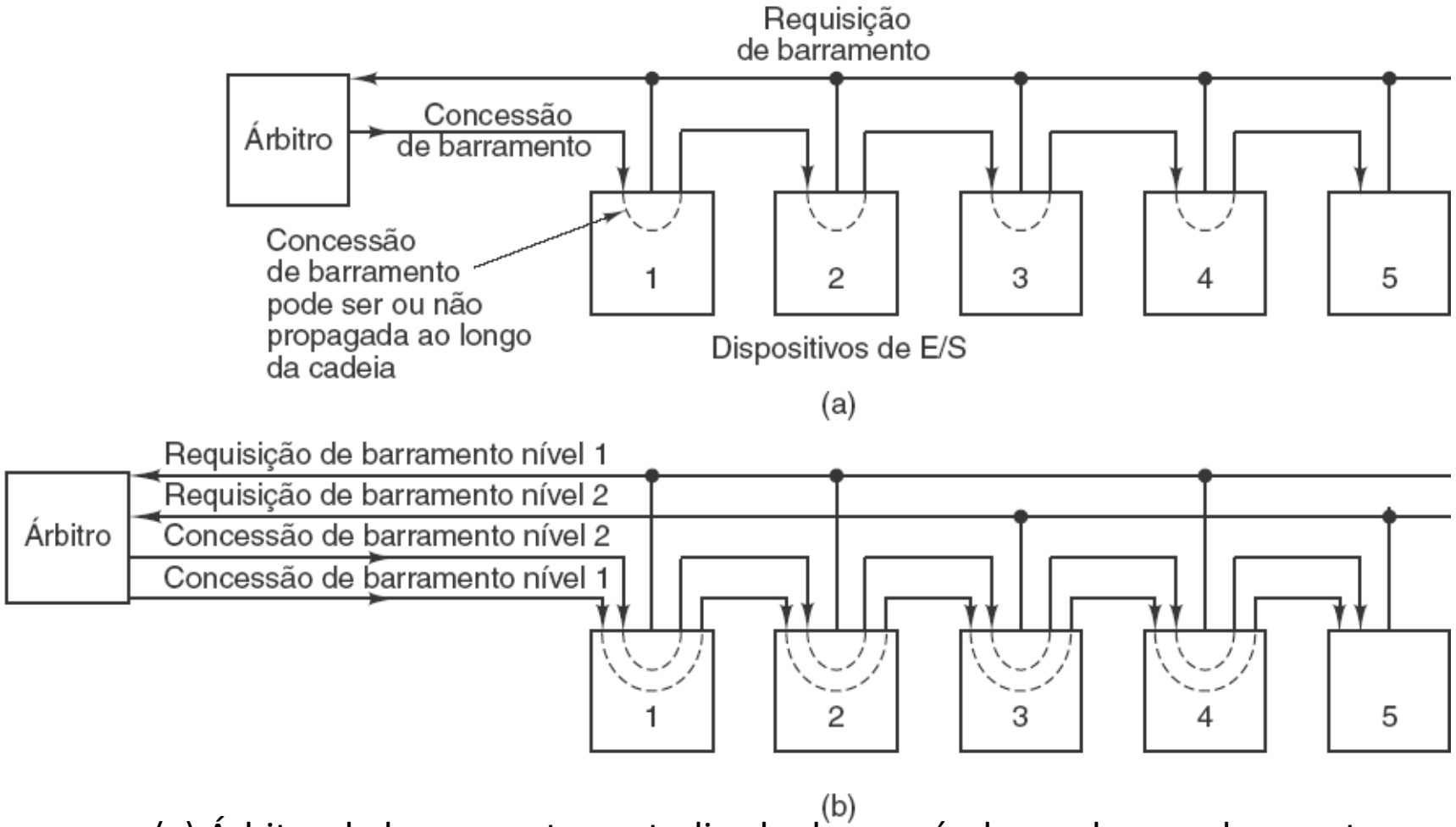
Transferência de bloco.

Operações de barramento



Utilização do controlador de interrupção 8259A.

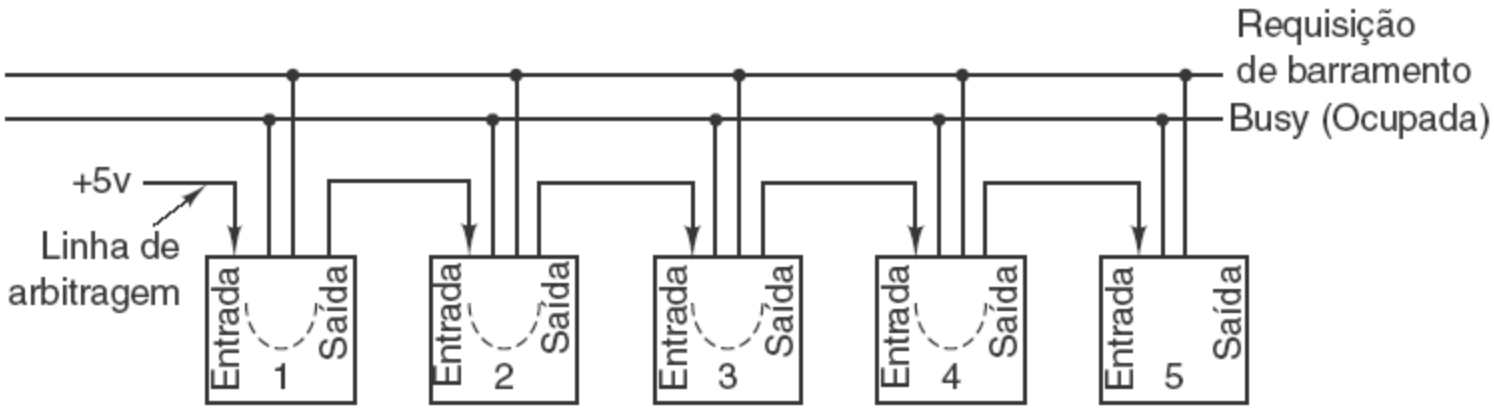
Arbitragem de barramento



(a) Árbitro de barramento centralizado de um nível usando encadeamento em série (daisy chaining).

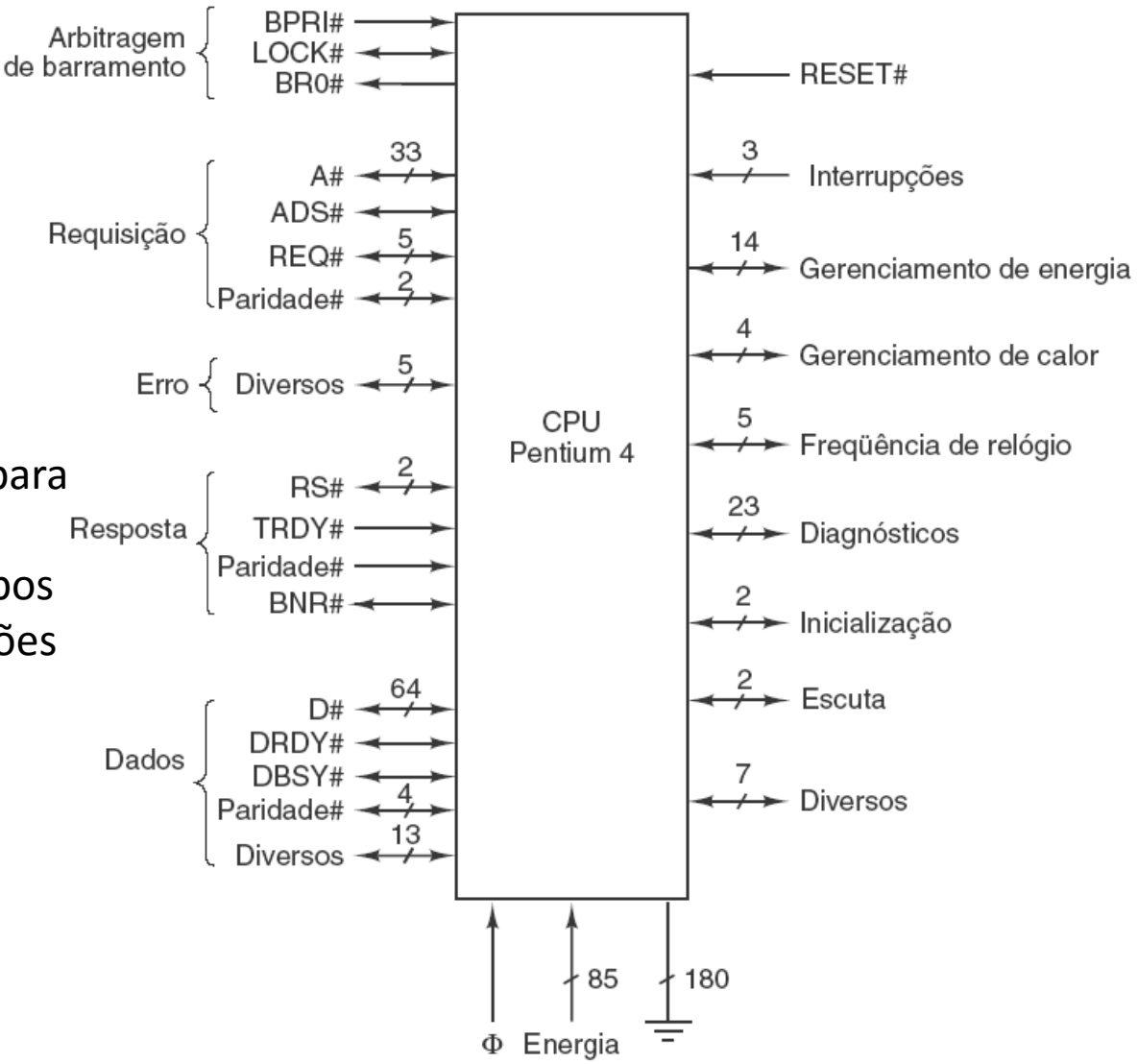
(b) O mesmo árbitro usando dois níveis.

Arbitragem de barramento (2)



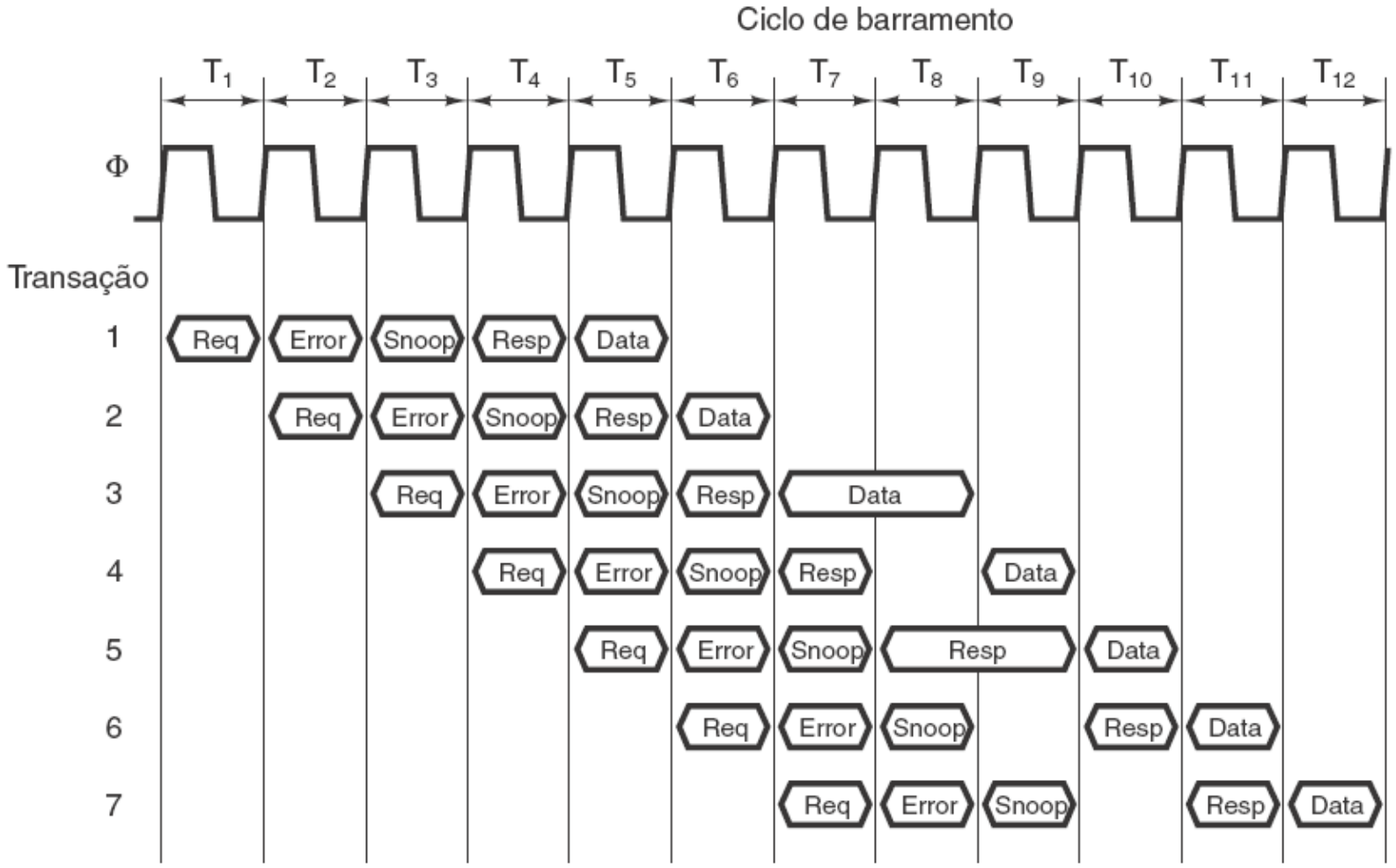
Arbitragem de barramento descentralizada.

Pinagem lógica do Pentium 4



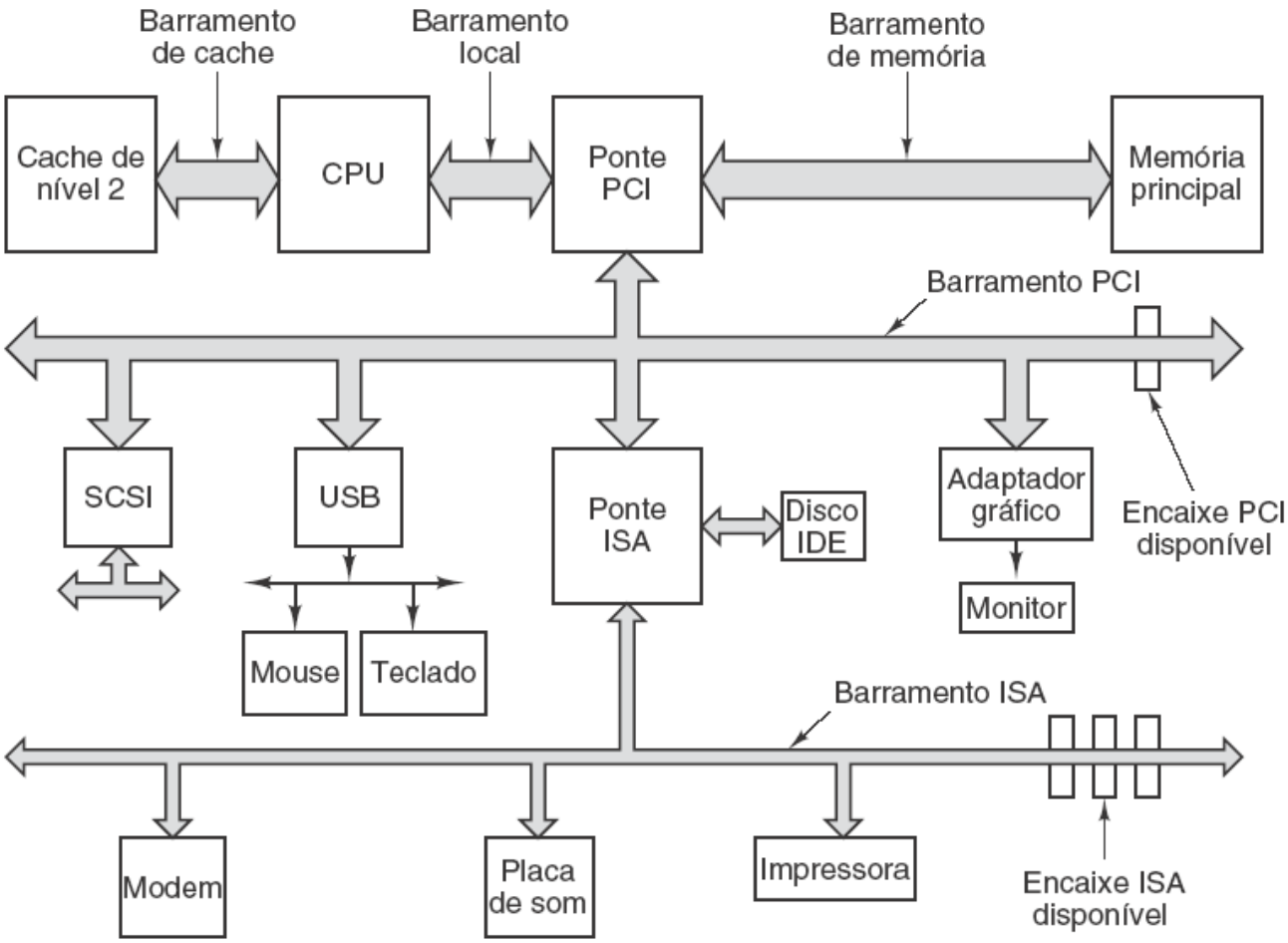
Pinagem lógica do Pentium 4. Nomes em letras maiúsculas são nomes oficiais usados pela Intel para sinais individuais. Nomes em maiúsculas e minúsculas são grupos de sinais relacionados ou descrições de sinais.

Paralelismo no barramento de memória do Pentium 4



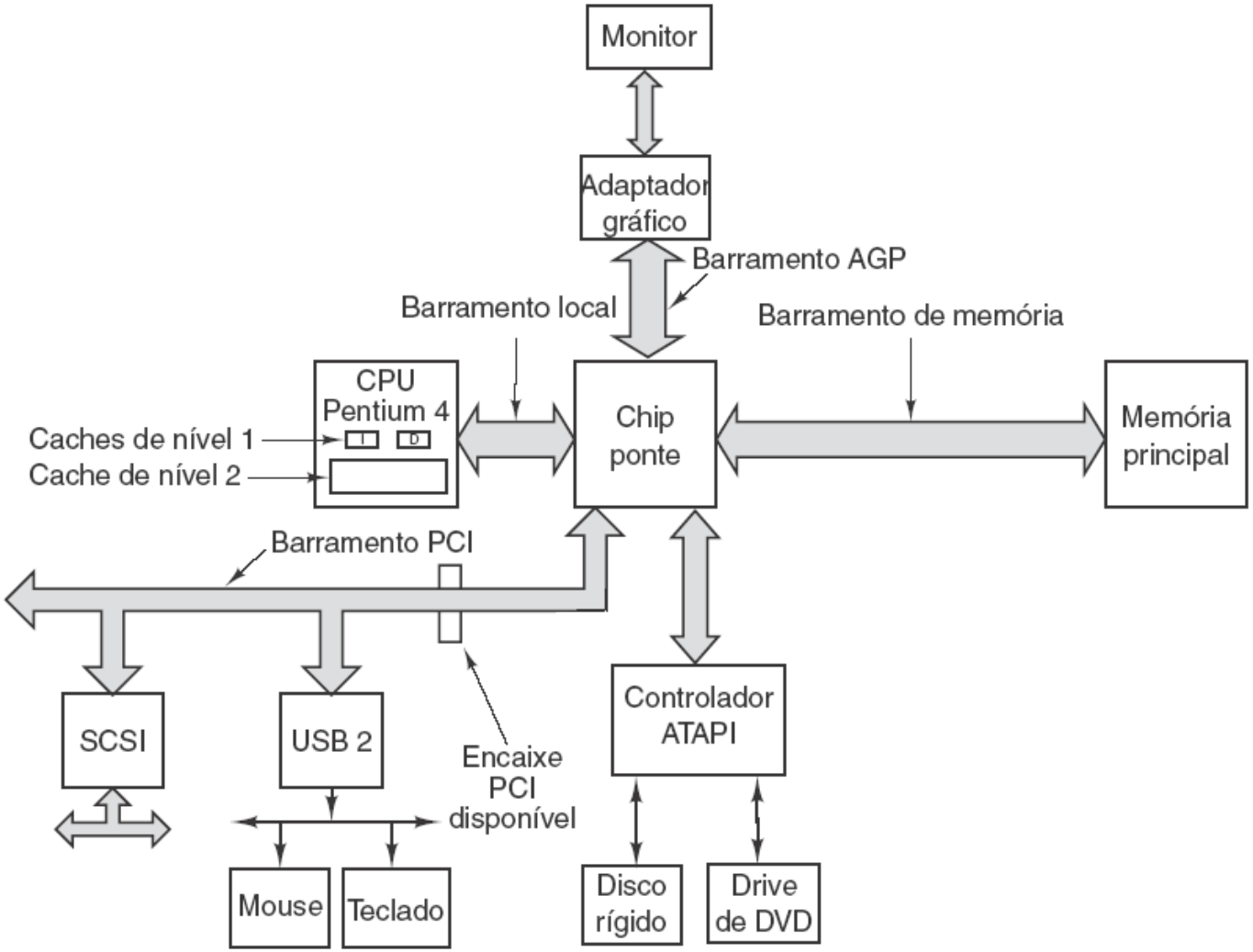
Requisições com paralelismo no barramento de memória do Pentium 4.

O barramento PCI



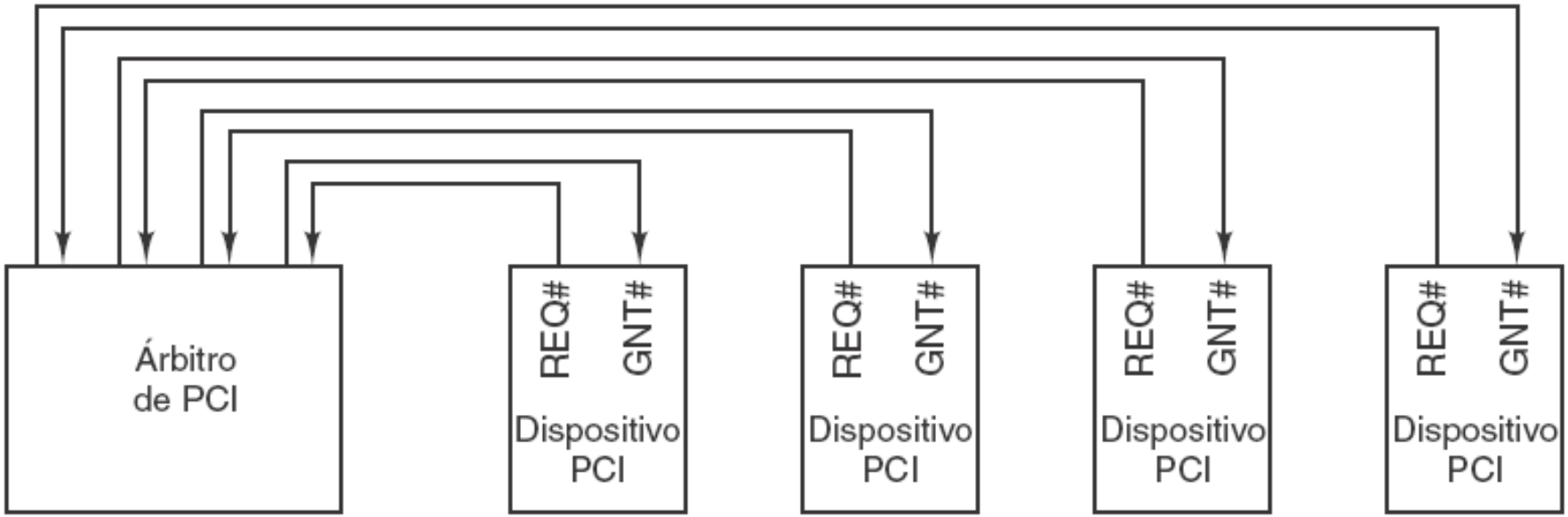
Arquitetura de um dos primeiros sistemas Pentium. Os barramentos representados por linhas mais largas têm mais largura de banda do que os representados por linhas mais finas, mas a figura não está em escala.

O barramento PCI



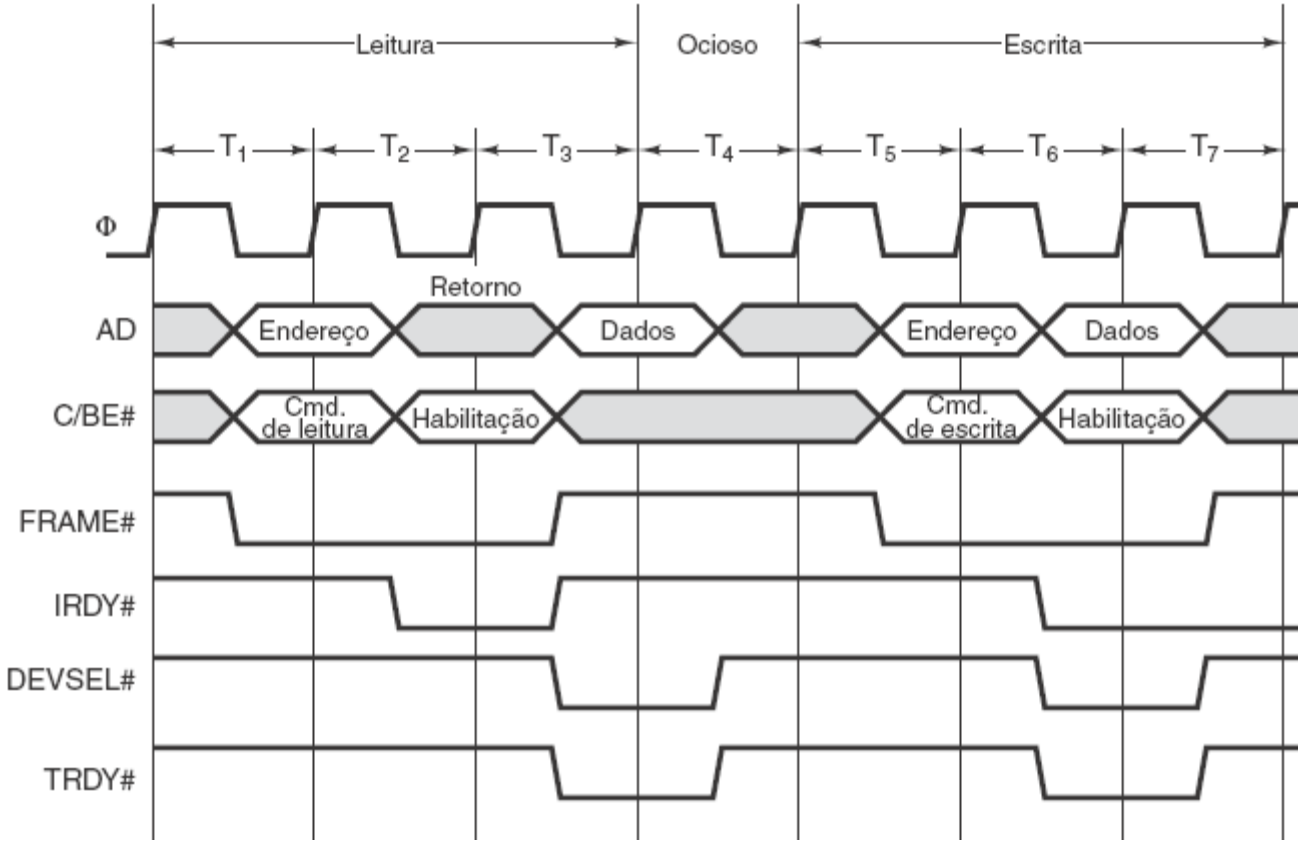
A estrutura do barramento de um Pentium 4 moderno.

Arbitragem de barramento PCI



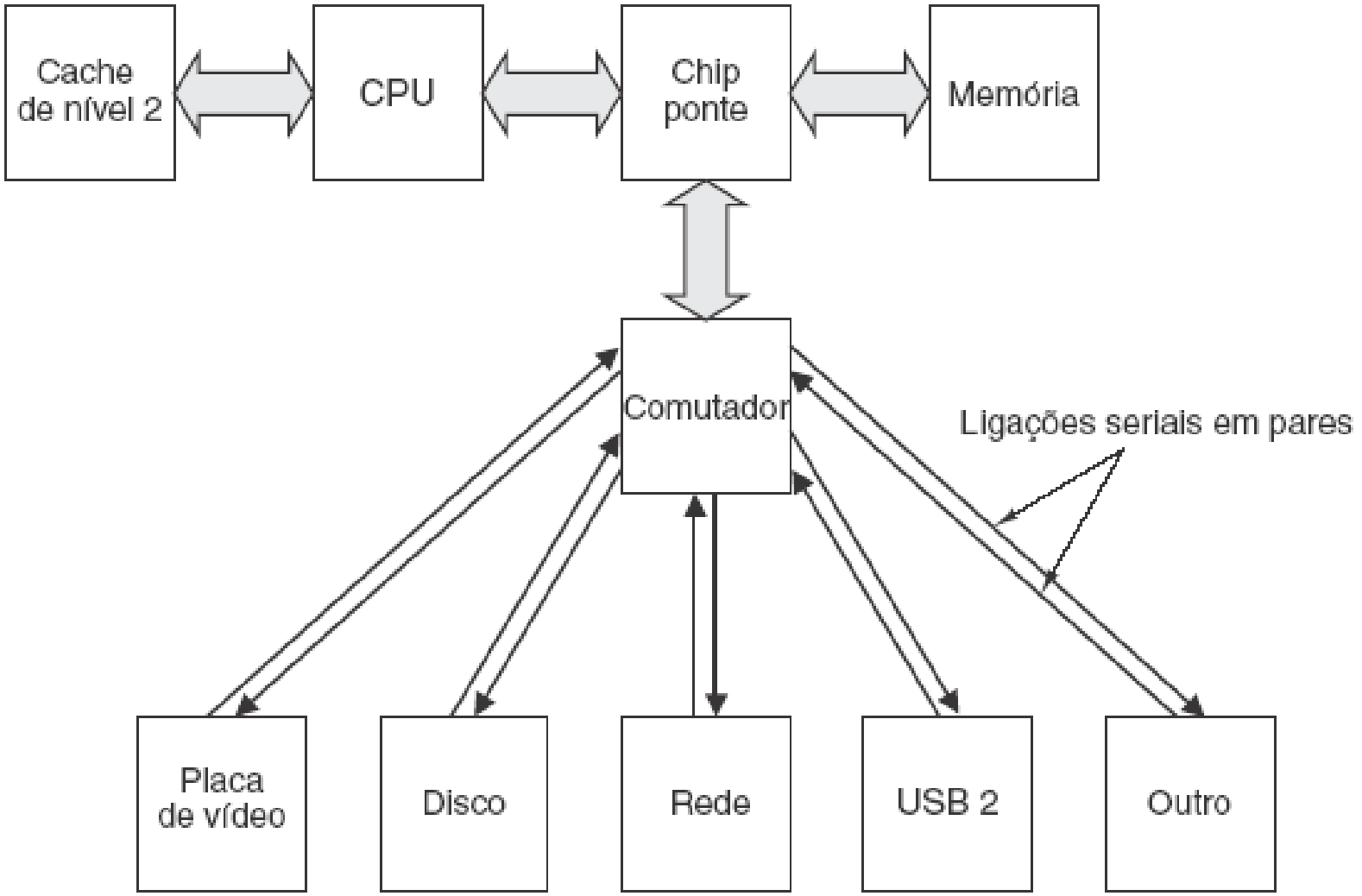
O barramento PCI usa um árbitro de barramento centralizado.

Transações no barramento PCI



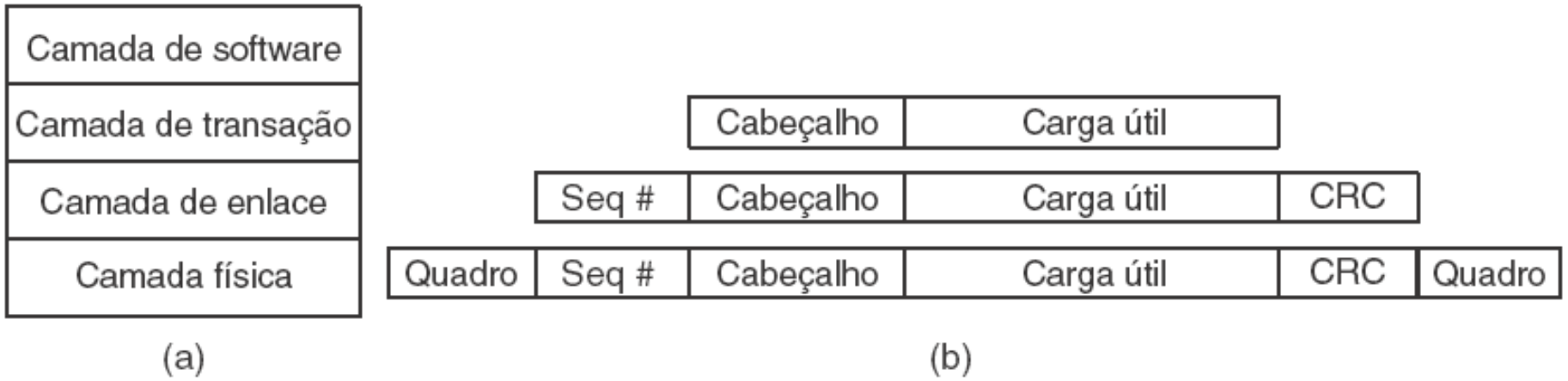
Exemplos de transações no barramento PCI de 32 bits. Os três primeiros ciclos são usados para uma operação de leitura, em seguida um ciclo ocioso e depois três ciclos para uma operação de escrita.

PCI Express



Sistema PCI Express.

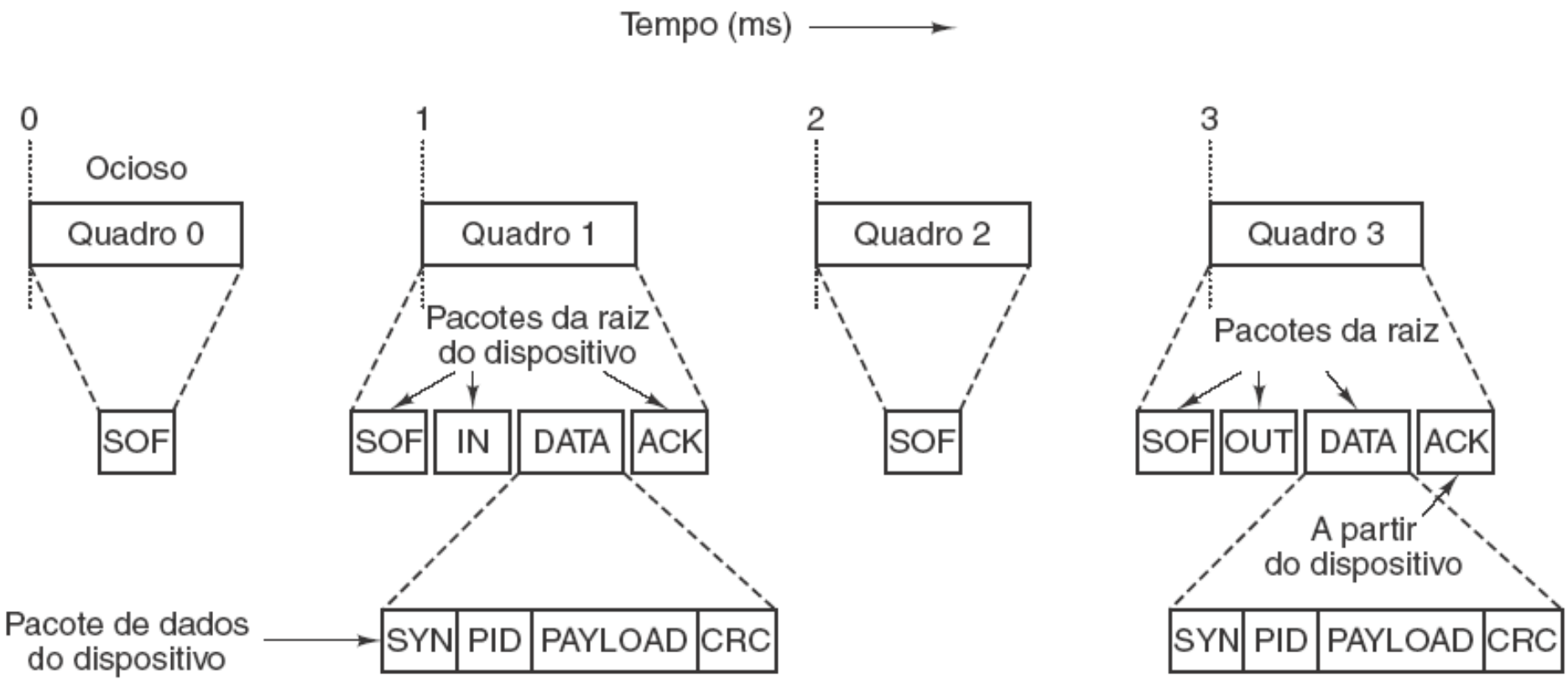
Pilha de protocolos do PCI Express



(a) Pilha de protocolos do PCI Express.

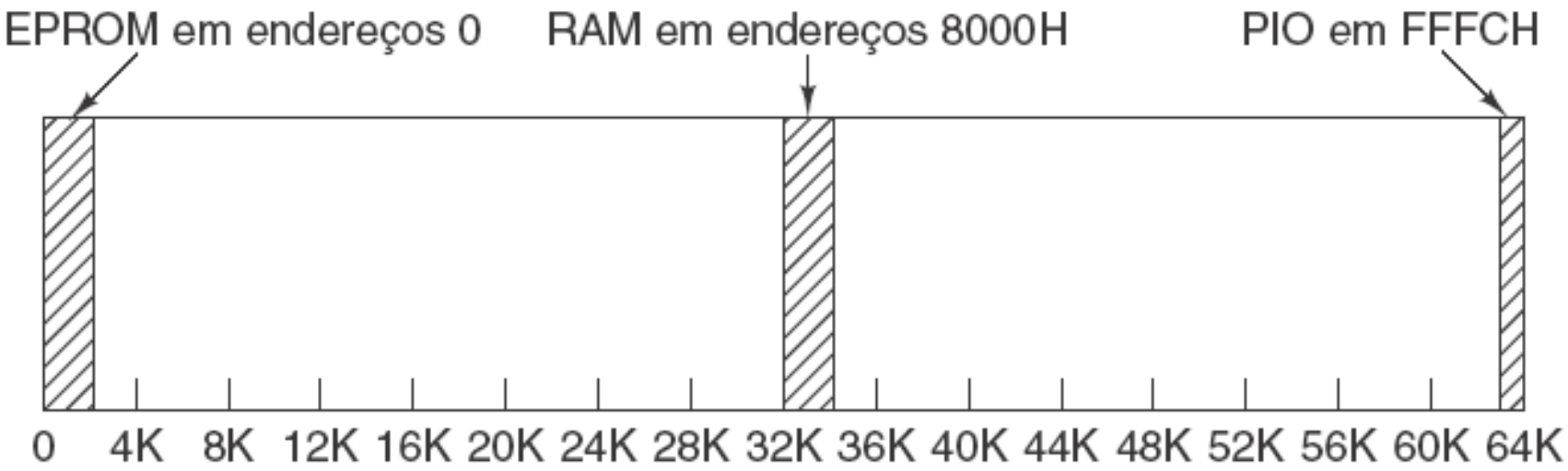
(b) Formato de um pacote.

Barramento Serial Universal



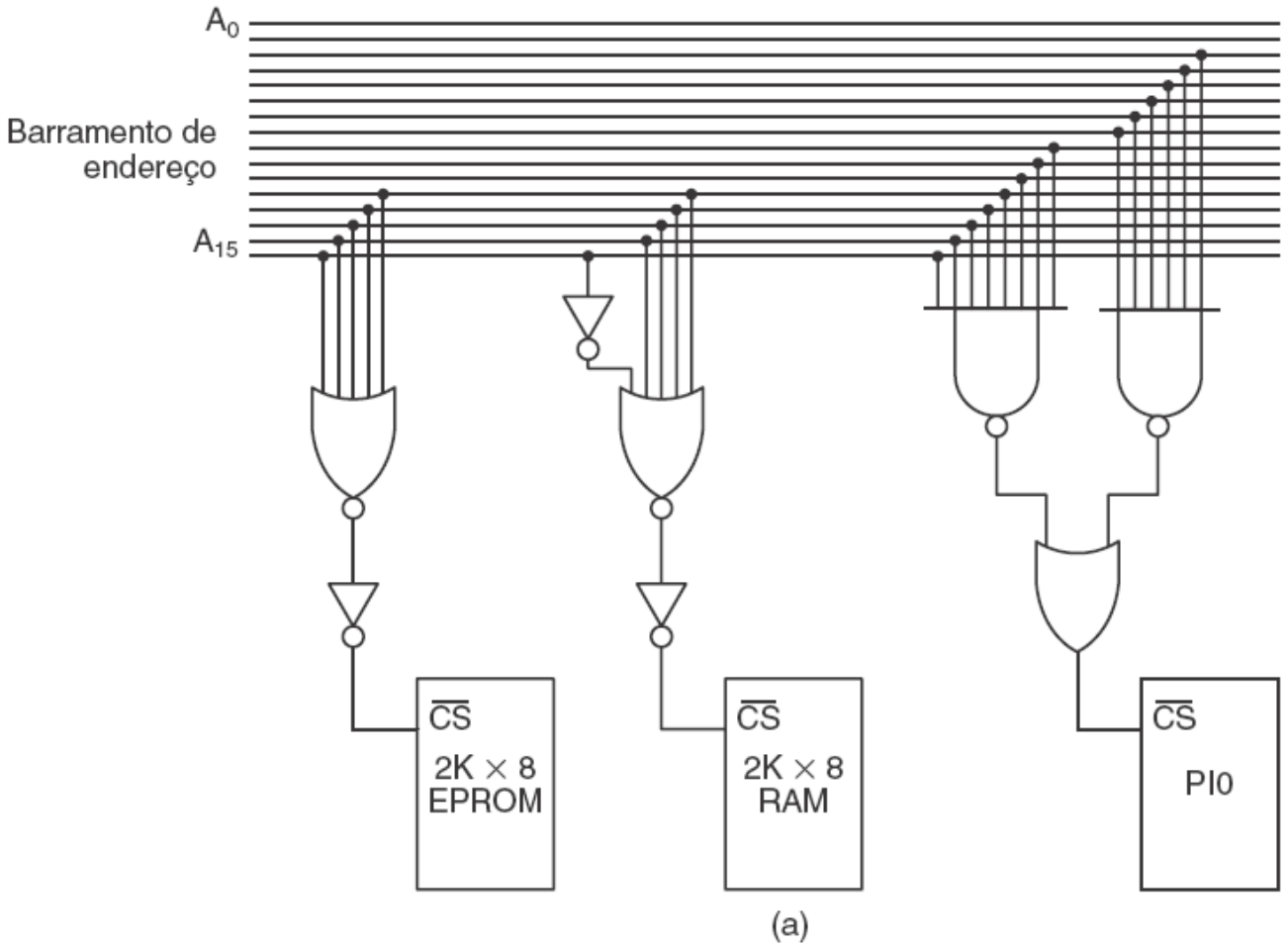
Hub-raiz USB envia quadros a cada 1,00 ms.

Decodificação de endereço



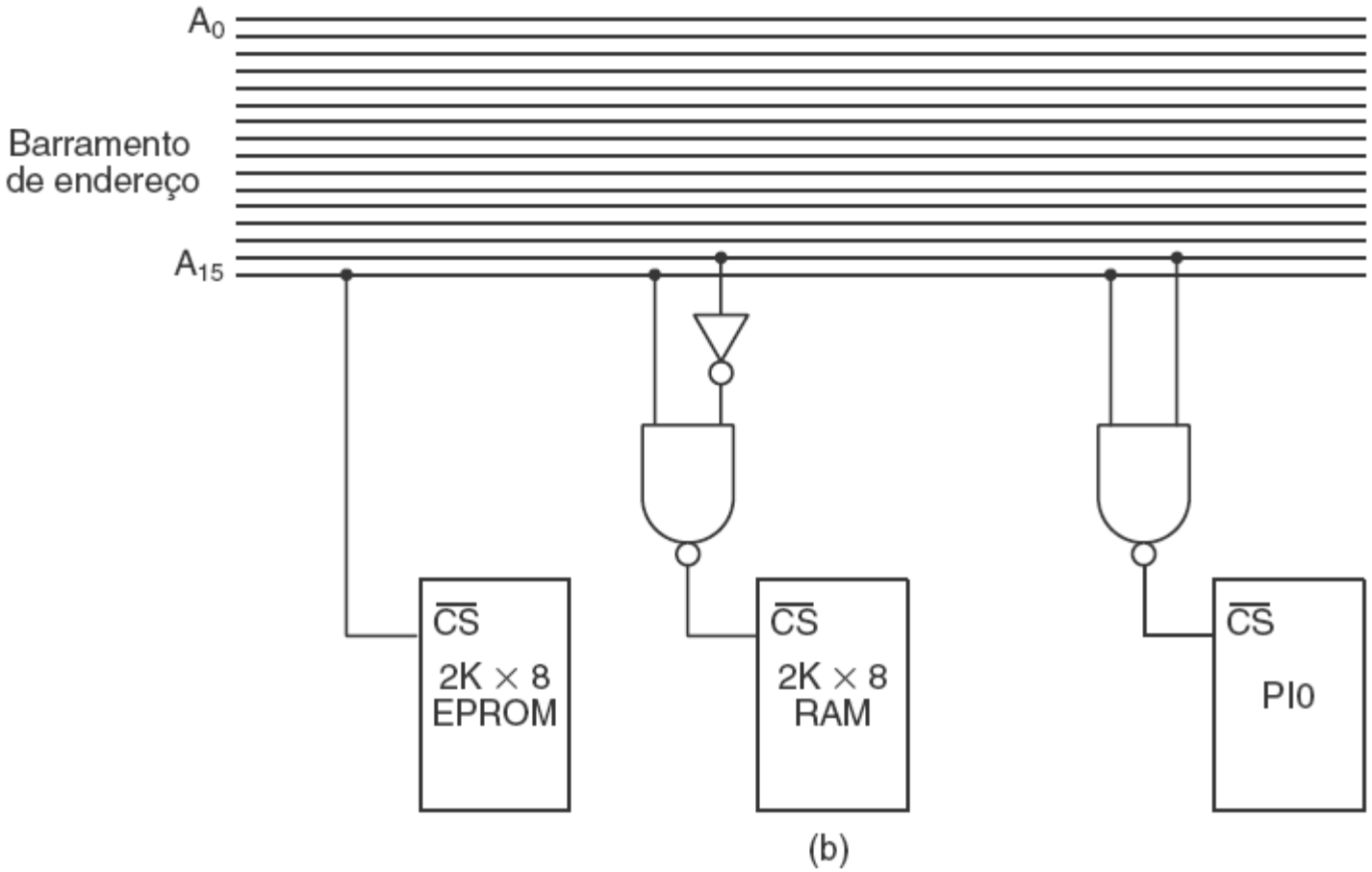
Localização de EPROM, RAM e PIO em nosso espaço de endereço de 64 KB.

Decodificação de endereço (2)



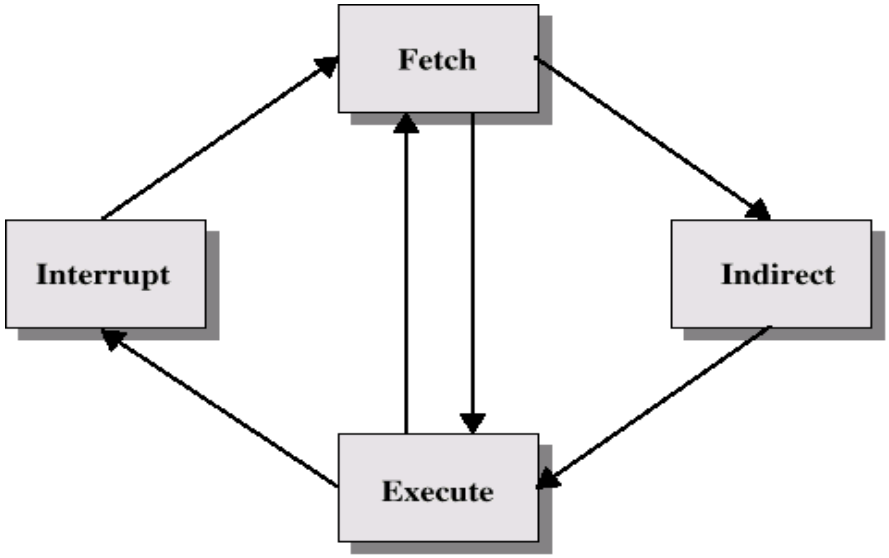
Decodificação de endereço completo.

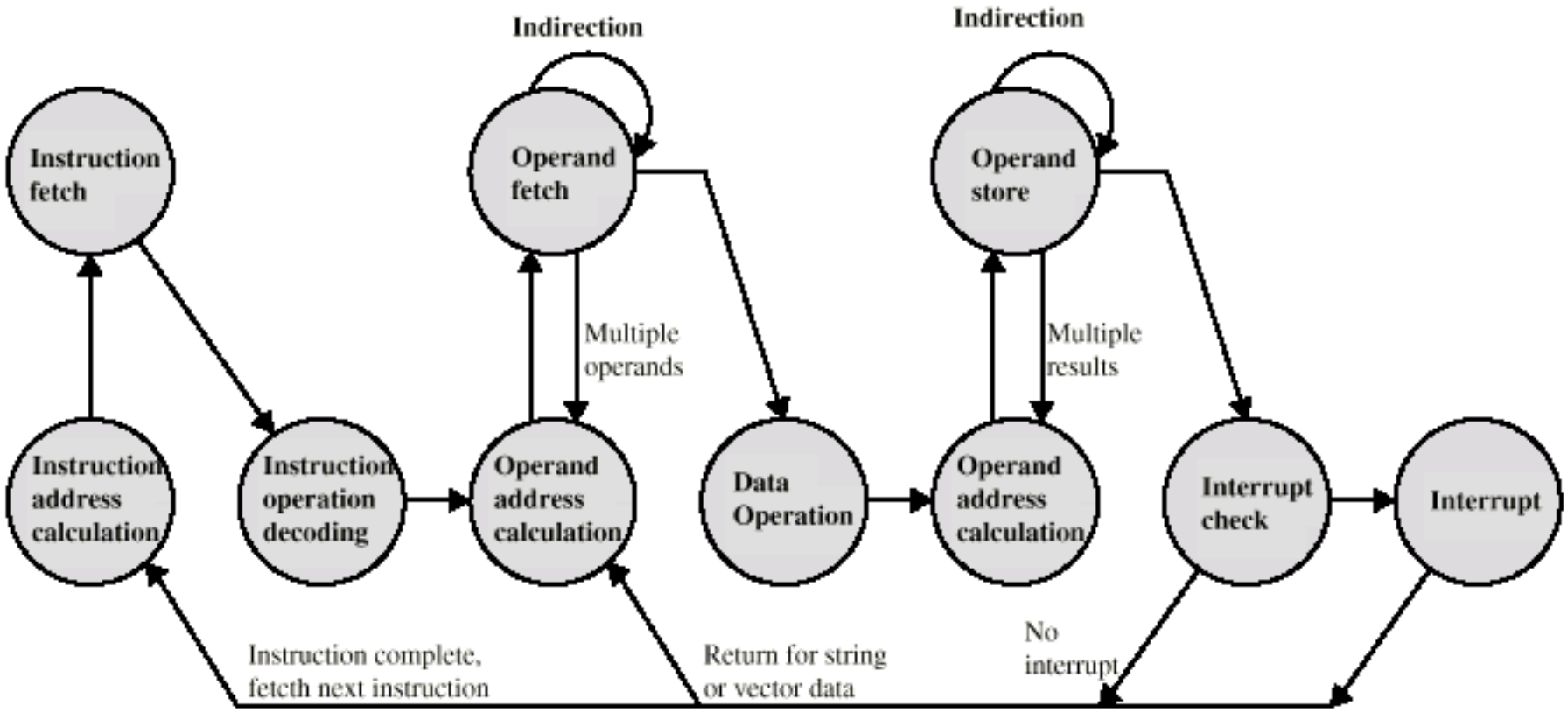
Decodificação de endereço (3)



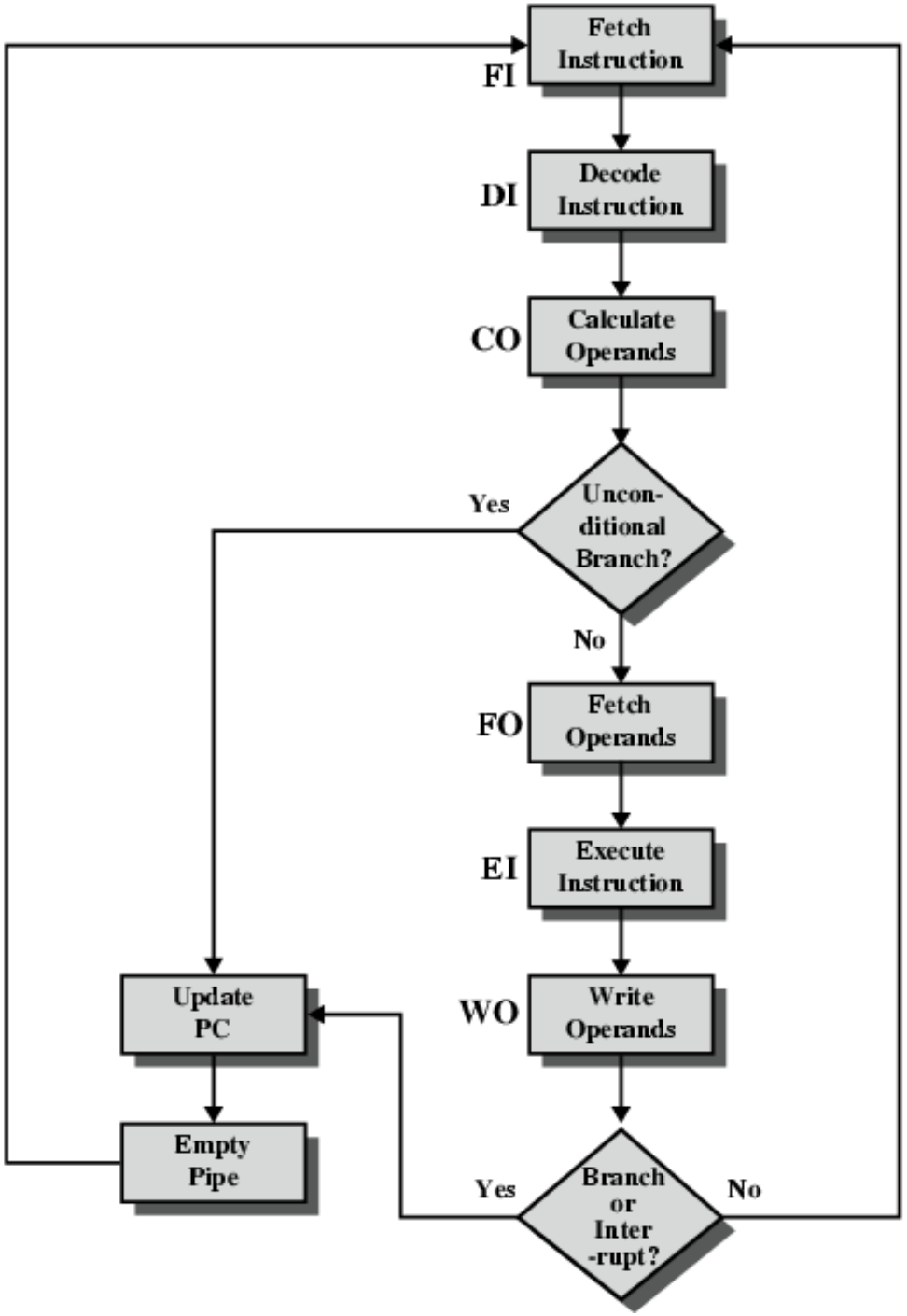
Decodificação parcial de endereço.

Instruction Cycle





Pipeline de Instrução de 6 estágios



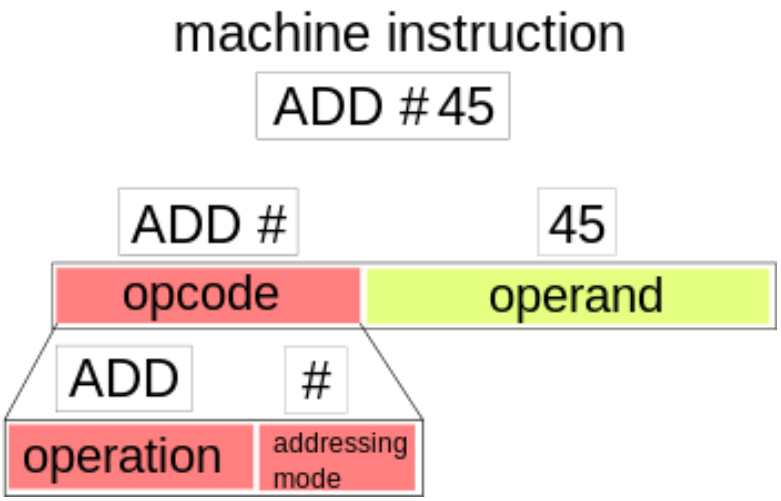
Linguagem de Máquina

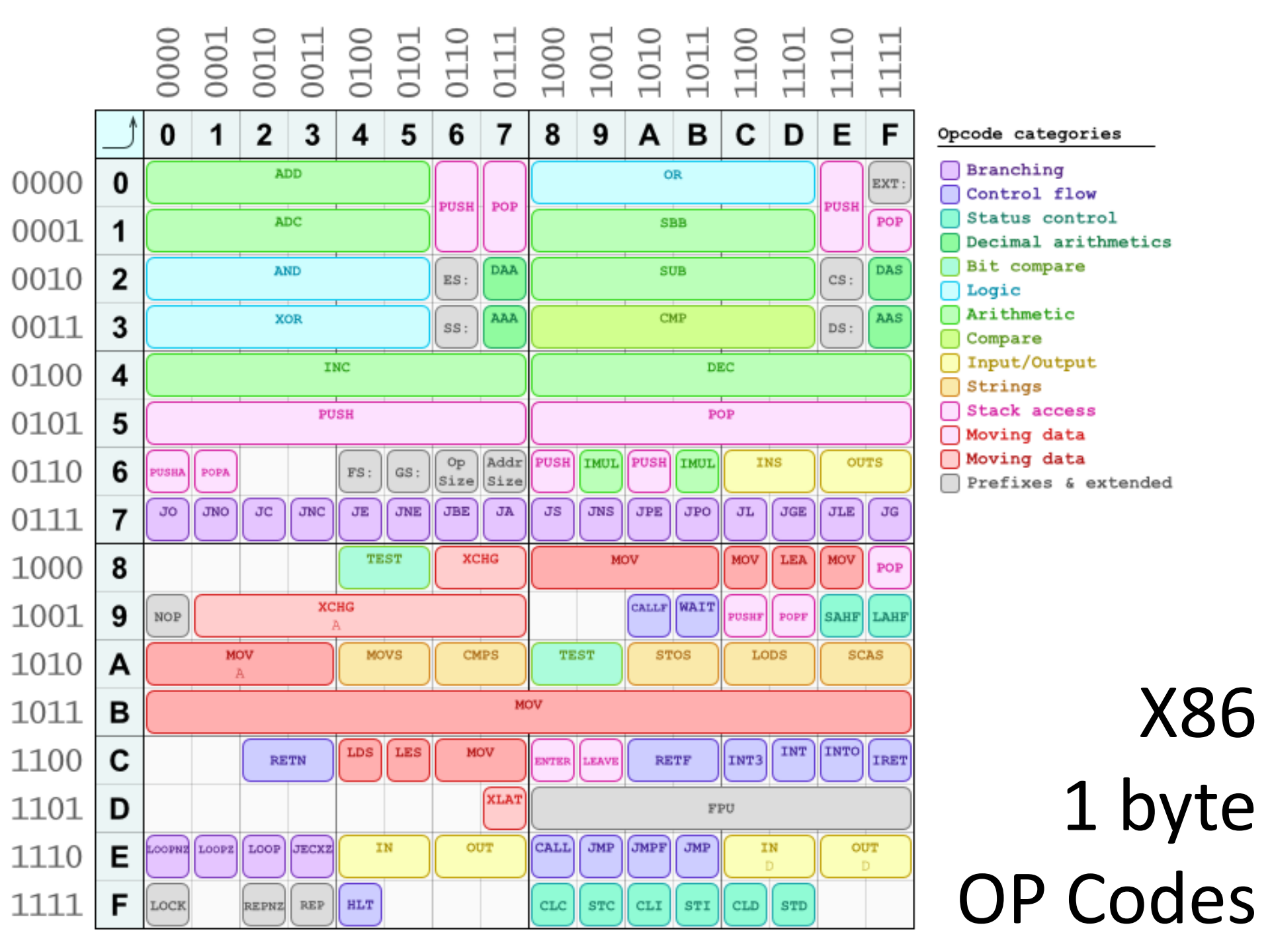
Linguagem de máquina

- Unidade de controle faz a leitura da próxima instrução da memória (*instruction pointer*)
- Cada operação tem um código exclusivo, uma sequência de 8 bits indicando a operação:
  - Ex. ADD, MOVE, JUMP, BREAK.
- Lê bloco de dados que segue a instrução, de acordo com o código dela
- Um segmento de hardware aceita o código e emite os sinais de controle para transferência de dados e acionamento da ULA

Instruction Set Archtecture – ISA

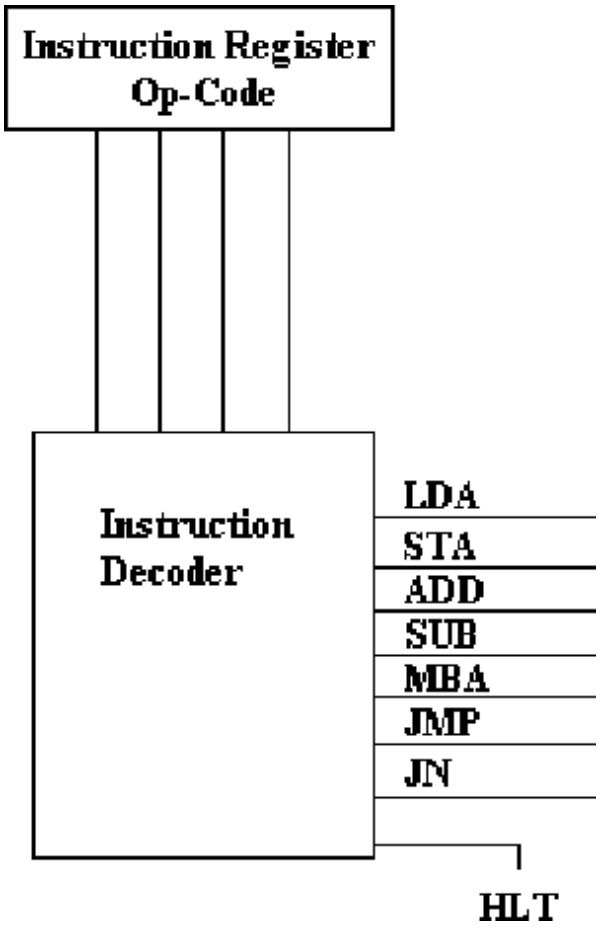
- Op code – Código da instrução (binário)
- Símbolo – nome da instrução (JMP, ADD, BRK)
- Address – posição de memória do dado
- Data – Entrada direta de dado (parâmetro)

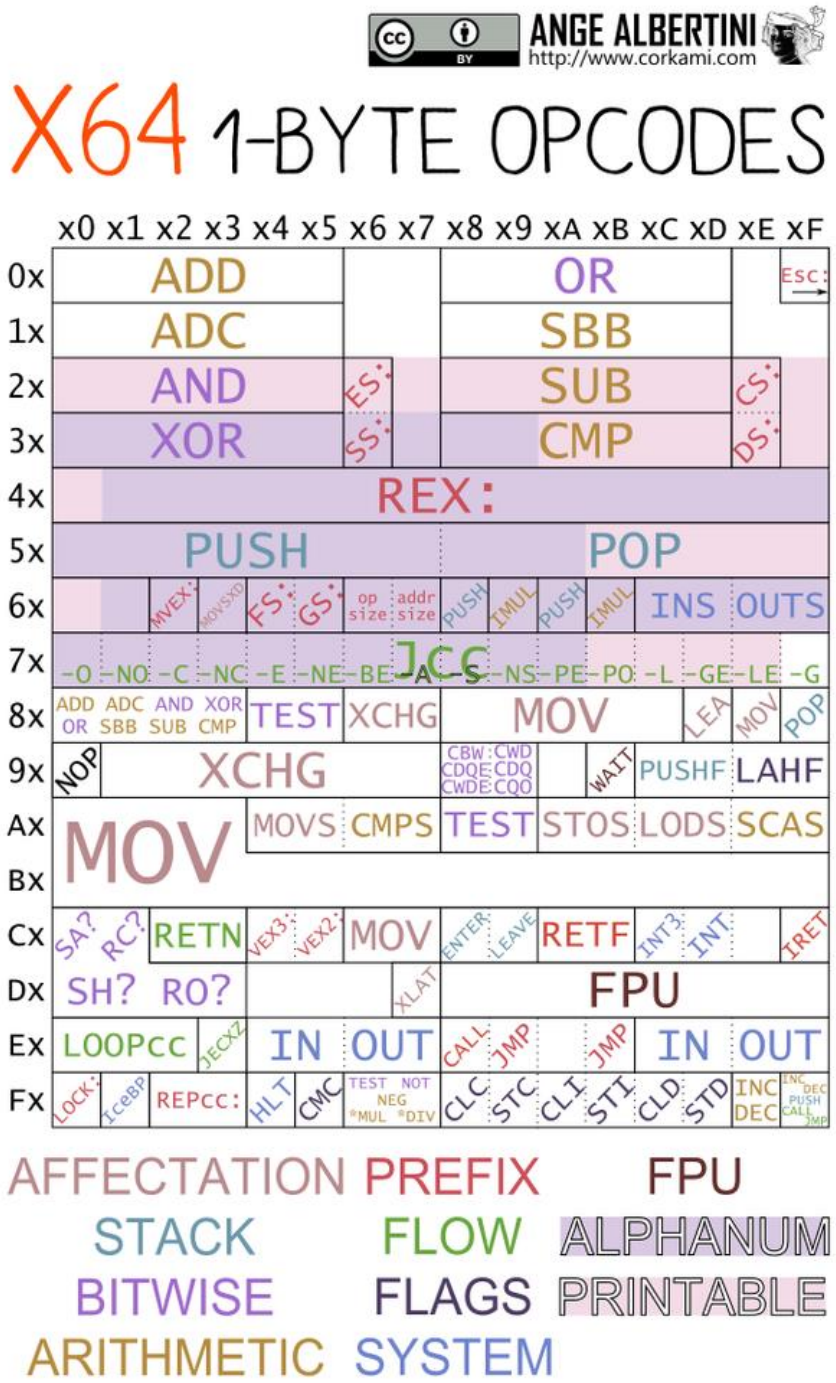




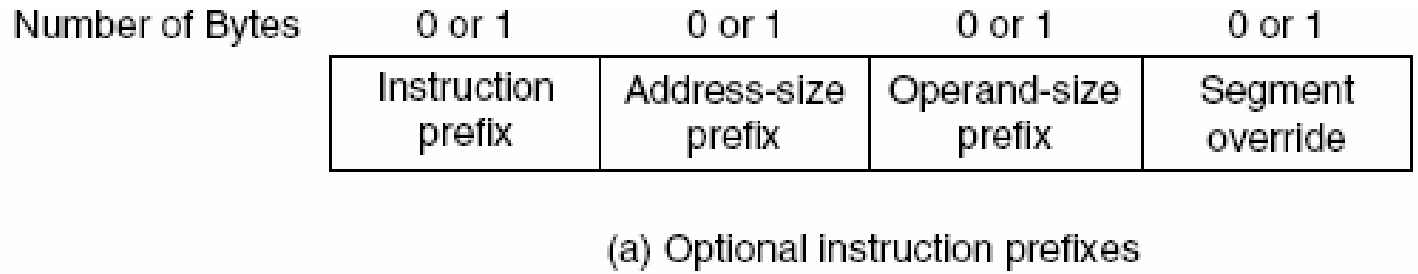
X86 1 byte OP Codes

X64 OP Codes

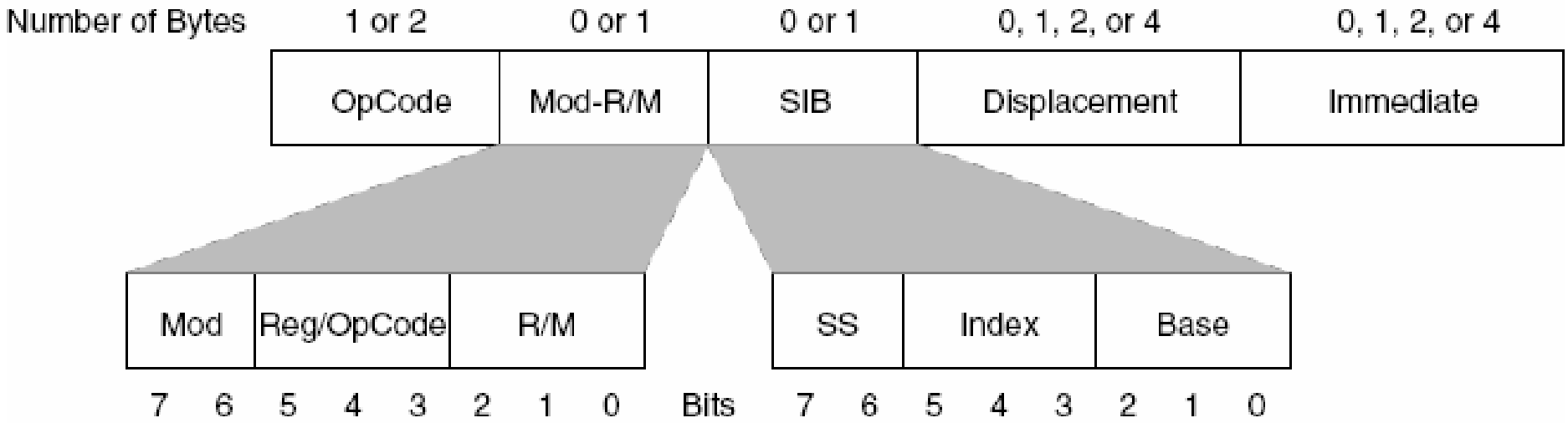

Instruction Register
Op-Code
Instruction
Decoder
LDA
STA
ADD
SUB
MBA
JMP
JN
HLT


ANGE ALBERTINI
http://www.corkami.com
X64 1-BYTE OPCODES
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 xA xB xC xD xE xF
0x ADD OR Esc:
1x ADC SBB
2x AND ES: SUB CS:
3x XOR SS: CMP DS:
4x REX:
5x PUSH POP
6x MVEX: MOVSXD FS: GS: op size addr size PUSH IMUL PUSH IMUL INS OUTS
7x JCC -O -NO -C -NC -E -NE -BE -A -S -NS -PE -PO -L -GE -LE -G
8x ADD ADC AND XOR OR SBB SUB CMP TEST XCHG MOV LEA MOV POP
9x NOP XCHG CBW: CWD CDQE: CDQ CWDE: CQO WAIT PUSHF LAHF
Ax MOV MOVS CMPS TEST STOS LODS SCAS
Bx
Cx SA? RC? RETN VEX3: VEX2: MOV ENTER LEAVE RETF INT3 INT IRET
Dx SH? RO? XLAT FPU
Ex LOOPcc JECXZ IN OUT CALL JMP JMP IN OUT
Fx LOCK: IcebP REPcc: HLT CMC TEST NOT NEG *MUL *DIV CLC STC CLI STI CLD STD INC DEC INC DEC PUSH CALL JMP
AFFECTATION PREFIX FPU
STACK FLOW ALPHANUM
BITWISE FLAGS PRINTABLE
ARITHMETIC SYSTEM

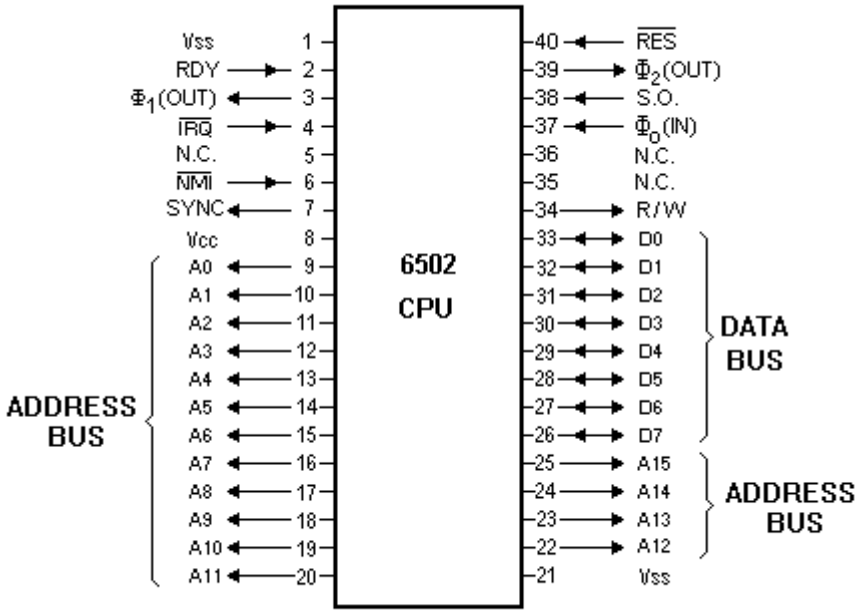
Estrutura de uma instrução

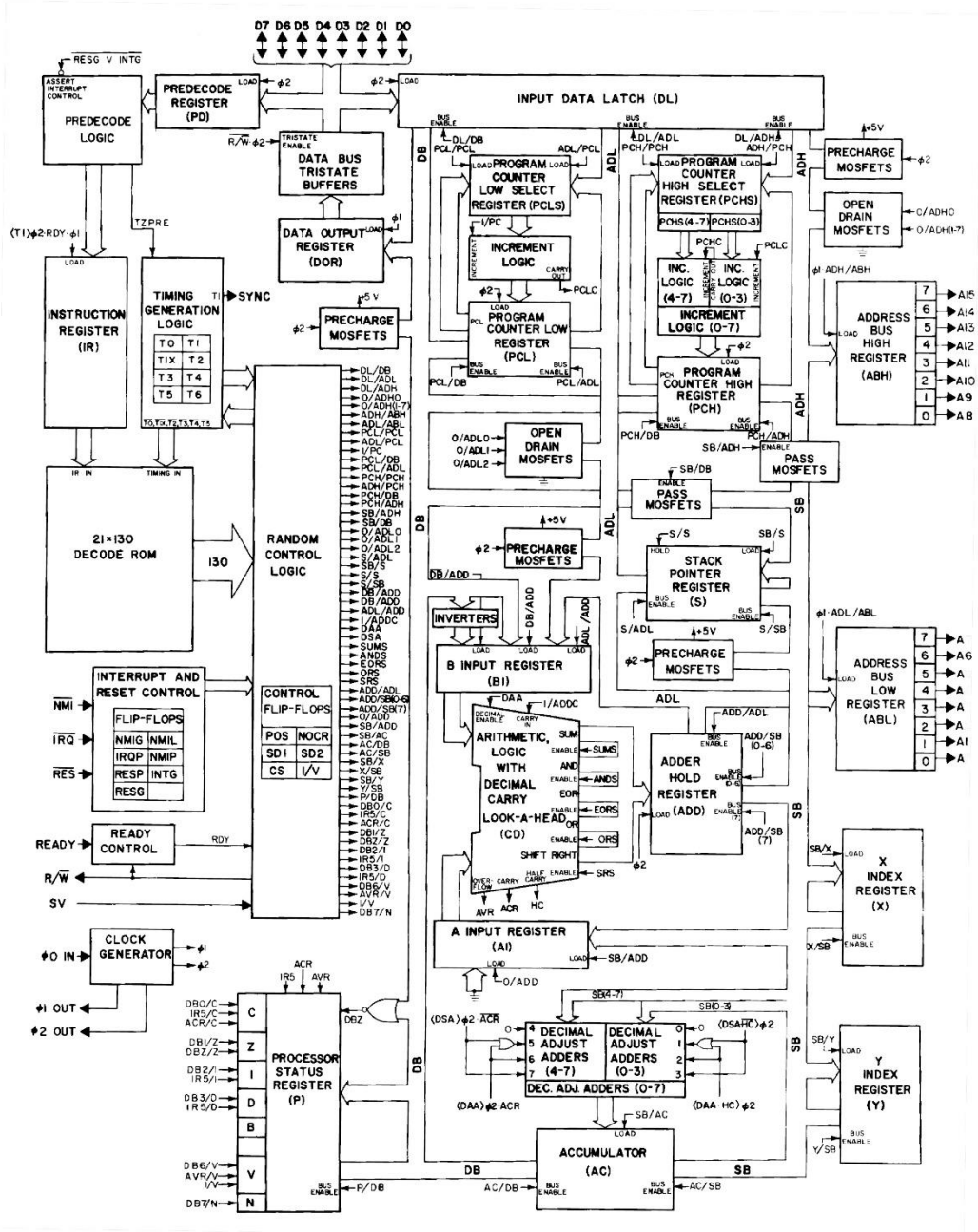


(a) Optional instruction prefixes



(b) General instruction format





http://porkrind.org/a2/
https://www.scullinsteel.com/apple2/

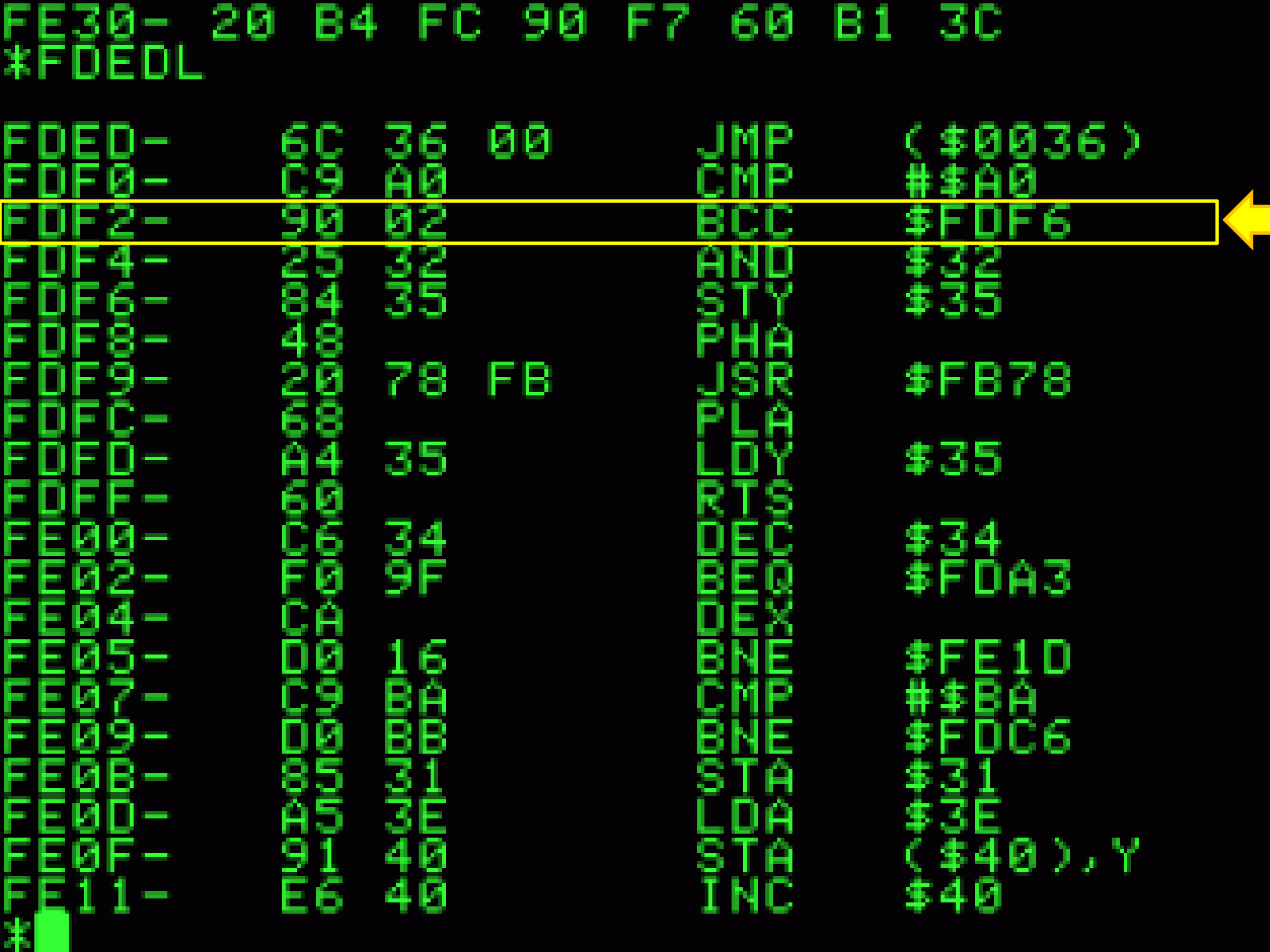
```
FE30- 20 B4 FC 90 F7 60 B1 3C
*FDEDL

FDED-    6C 36 00     JMP   ($0036)
FDF0-    C9 A0        CMP   #$A0
FDF2-    90 02        BCC   $FDF6
FDF4-    25 32        AND   $32
FDF6-    84 35        STY   $35
FDF8-    48           PHA
FDF9-    20 78 FB     JSR   $FB78
FDFC-    68           PLA
FDFD-    A4 35        LDY   $35
FDFF-    60           RTS
FE00-    C6 34        DEC   $34
FE02-    F0 9F        BEQ   $FDA3
FE04-    CA           DEX
FE05-    D0 16        BNE   $FE1D
FE07-    C9 BA        CMP   #$BA
FE09-    D0 BB        BNE   $FDC6
FE0B-    85 31        STA   $31
FE0D-    A5 3E        LDA   $3E
FE0F-    91 40        STA   ($40),Y
FE11-    E6 40        INC   $40
*
```

```
FE30- 20 B4 FC 90 F7 60 B1 3C
*FDEDL
```

PROGRAM COUNTER OU INSTRUCTION POINTER

```
FDED-    6C 36 00     JMP   ($0036)
FDF0-    C9 A0        CMP   #$A0
FDF2-    90 02        BCC   $FDF6
FDF4-    25 32        AND   $32
FDF6-    84 35        STY   $35
FDF8-    48           PHA
FDF9-    20 78 FB     JSR   $FB78
FDFC-    68           PLA
FDFD-    A4 35        LDY   $35
FDFF-    60           RTS
FE00-    C6 34        DEC   $34
FE02-    F0 9F        BEQ   $FDA3
FE04-    CA           DEX
FE05-    D0 16        BNE   $FE1D
FE07-    C9 BA        CMP   #$BA
FE09-    D0 BB        BNE   $FDC6
FE0B-    85 31        STA   $31
FE0D-    A5 3E        LDA   $3E
FE0F-    91 40        STA   ($40),Y
FE11-    E6 40        INC   $40
*
```


FE30- 20 B4 FC 90 F7 60 B1 3C
*FDEDL
FDED- 6C 36 00 JMP ($0036)
FDF0- C9 A0 CMP #$A0
FDF2- 90 02 BCC $FDF6
FDF4- 25 32 AND $32
FDF6- 84 35 STY $35
FDF8- 48 PHA
FDF9- 20 78 FB JSR $FB78
FDFC- 68 PLA
FDFD- A4 35 LDY $35
FDFF- 60 RTS
FE00- C6 34 DEC $34
FE02- F0 9F BEQ $FDA3
FE04- CA DEX
FE05- D0 16 BNE $FE1D
FE07- C9 BA CMP #$BA
FE09- D0 BB BNE $FDC6
FE0B- 85 31 STA $31
FE0D- A5 3E LDA $3E
FE0F- 91 40 STA ($40),Y
FE11- E6 40 INC $40
*

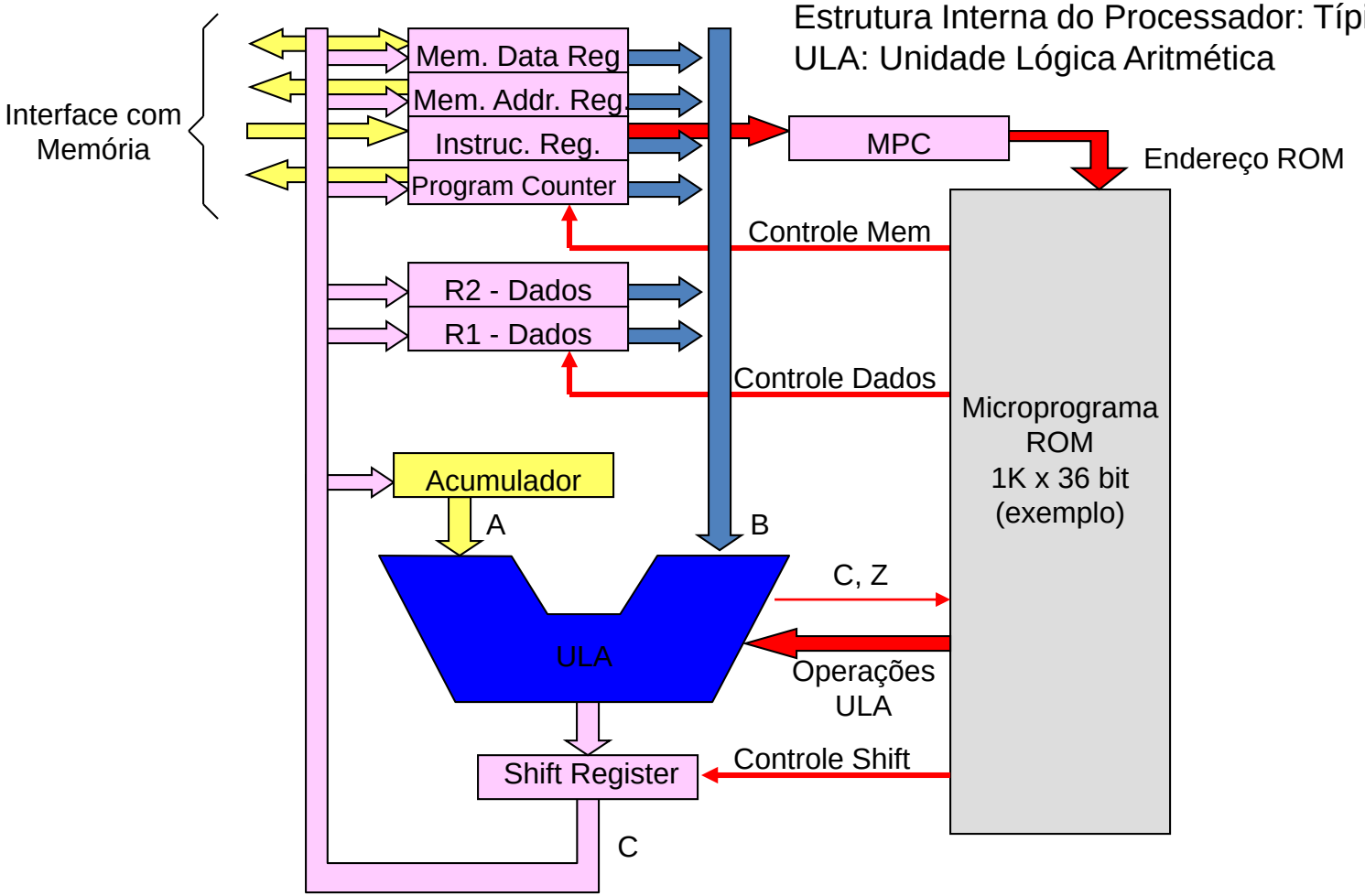
```
FE30- 20 B4 FC 90 F7 60 B1 3C
*FDEDL

FDED-    6C 36 00     JMP   ($0036)
FDF0-    C9 A0        CMP   #$A0
FDF2-    90 02        BCC   $FDF6
FDF4-    25 32        AND   $32
FDF6-    84 35        STY   $35
FDF8-    48           PHA
FDF9-    20 78 FB     JSR   $FB78
FDFC-    68           PLA
FDFD-    A4 35        LDY   $35
FDFF-    60           RTS
FE00-    C6 34        DEC   $34
FE02-    F0 9F        BEQ   $FDA3
FE04-    CA           DEX
FE05-    D0 16        BNE   $FE1D
FE07-    C9 BA        CMP   #$BA
FE09-    D0 BB        BNE   $FDC6
FE0B-    85 31        STA   $31
FE0D-    A5 3E        LDA   $3E
FE0F-    91 40        STA   ($40),Y
FE11-    E6 40        INC   $40
*
```

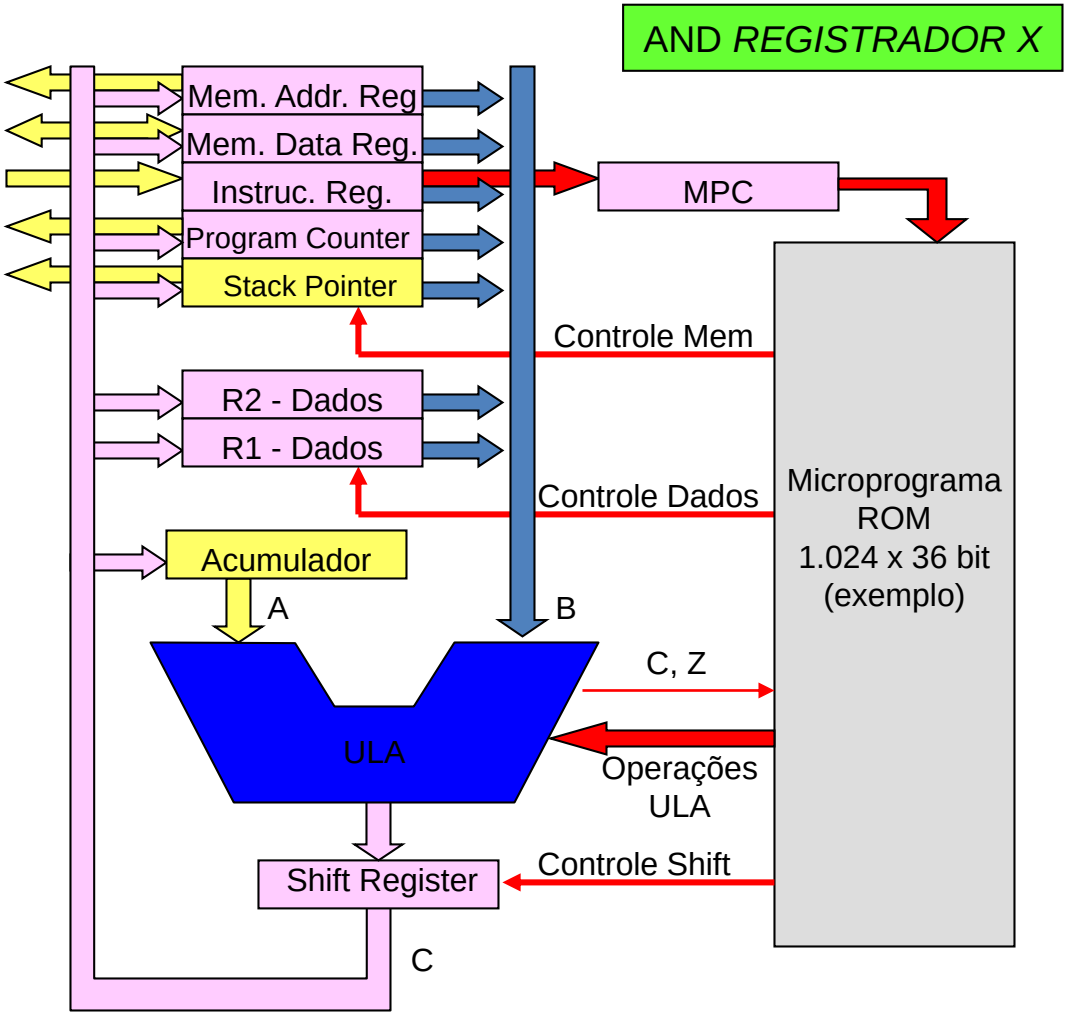
Tipos de Instruções

UCP: UNIDADE CENTRAL DE PROCESSAMENTO

UCP Didático de 8 bits:
Estrutura Interna do Processador: Típica von Neumann
ULA: Unidade Lógica Aritmética



INSTRUÇÕES DE UM BYTE



Exemplo:

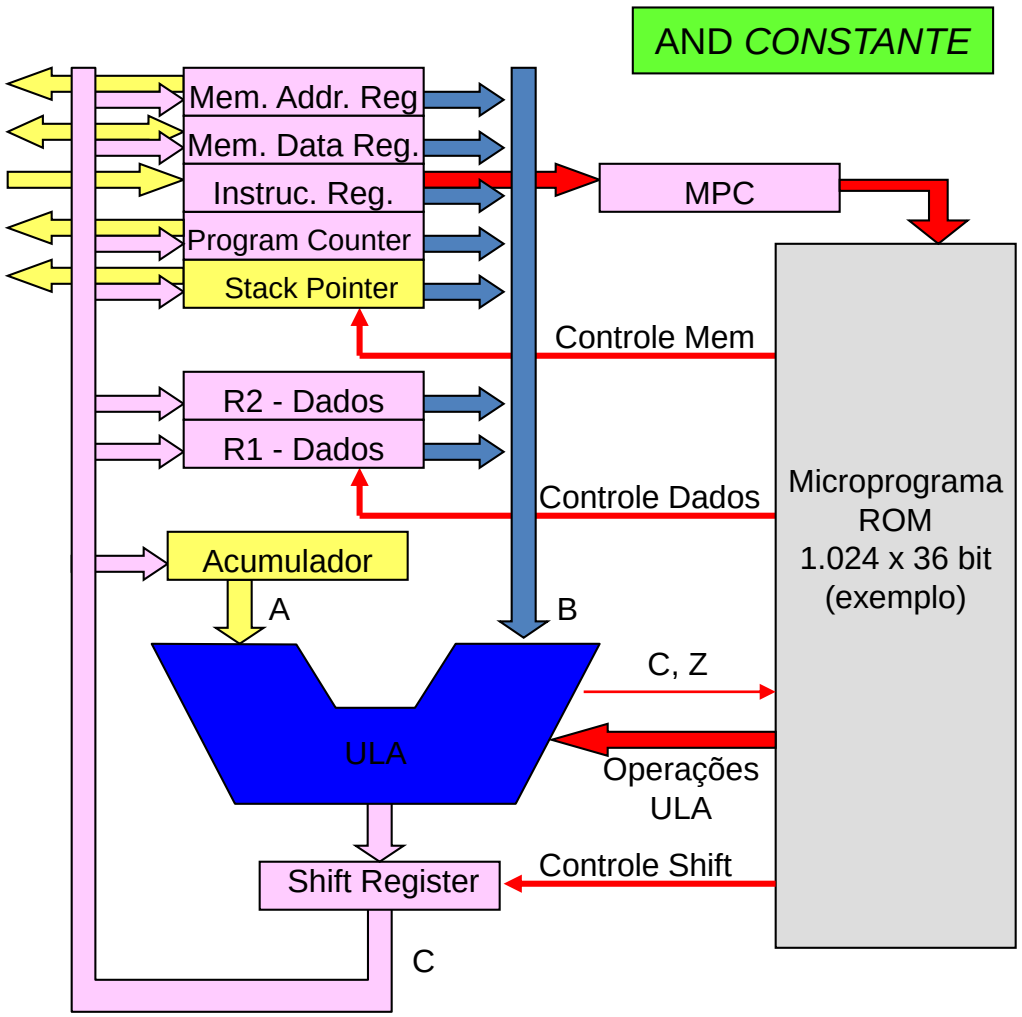
AND ACC, R1: (ACC ← ACC *AND* R1)

1: FETCH INSTRUÇÃO
Q1: PC → Via de Endereços
Q2: Comando de Leitura
Q3: Memória → Via de Dados
Q4: Via de Dados → IR

2: Acumulador = Acumulador AND R1
Q1: R1 → Barramento B
Q2: ULA: Operação AND
Q3: ULA: Retenção Barramento A
Q4: Barramento C → Acumulador

3: PC = PC + 1
Q1: ULA: Bloqueia Barramento A
Q2: PC → Barramento B
Q3: ULA: Incrementa Barramento B
Q4: Barramento C → PC

INSTRUÇÕES DE DOIS BYTES



Exemplo:

AND ACC, 5A: (ACC ← ACC *AND* 5A)

1: FETCH INSTRUÇÃO

2: PC = PC + 1

3: FETCH CONSTANTE

4: Acum. = Acum. AND Constante

5: PC = PC + 1

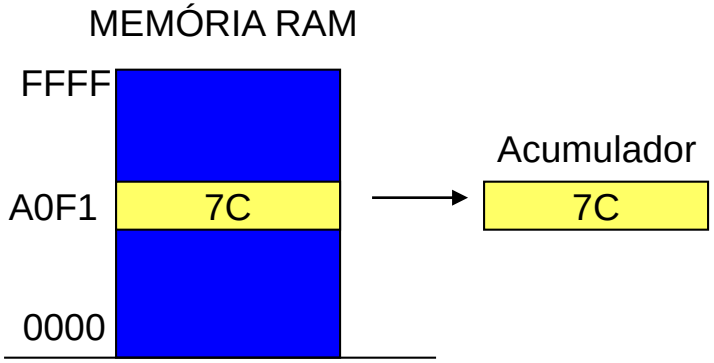
Exemplo: http://www.6502.org/tutorials/6502opcodes.html

INSTRUÇÕES DE TRÊS BYTES - 1

LEITURA DE UM DADO DA MEMÓRIA RAM: INSTRUÇÃO LOAD

Especificar o endereço de 16 bits da posição da memória a ser lida

Exemplo: leitura da posição A0F1, que contém o valor 7C



Exemplo:

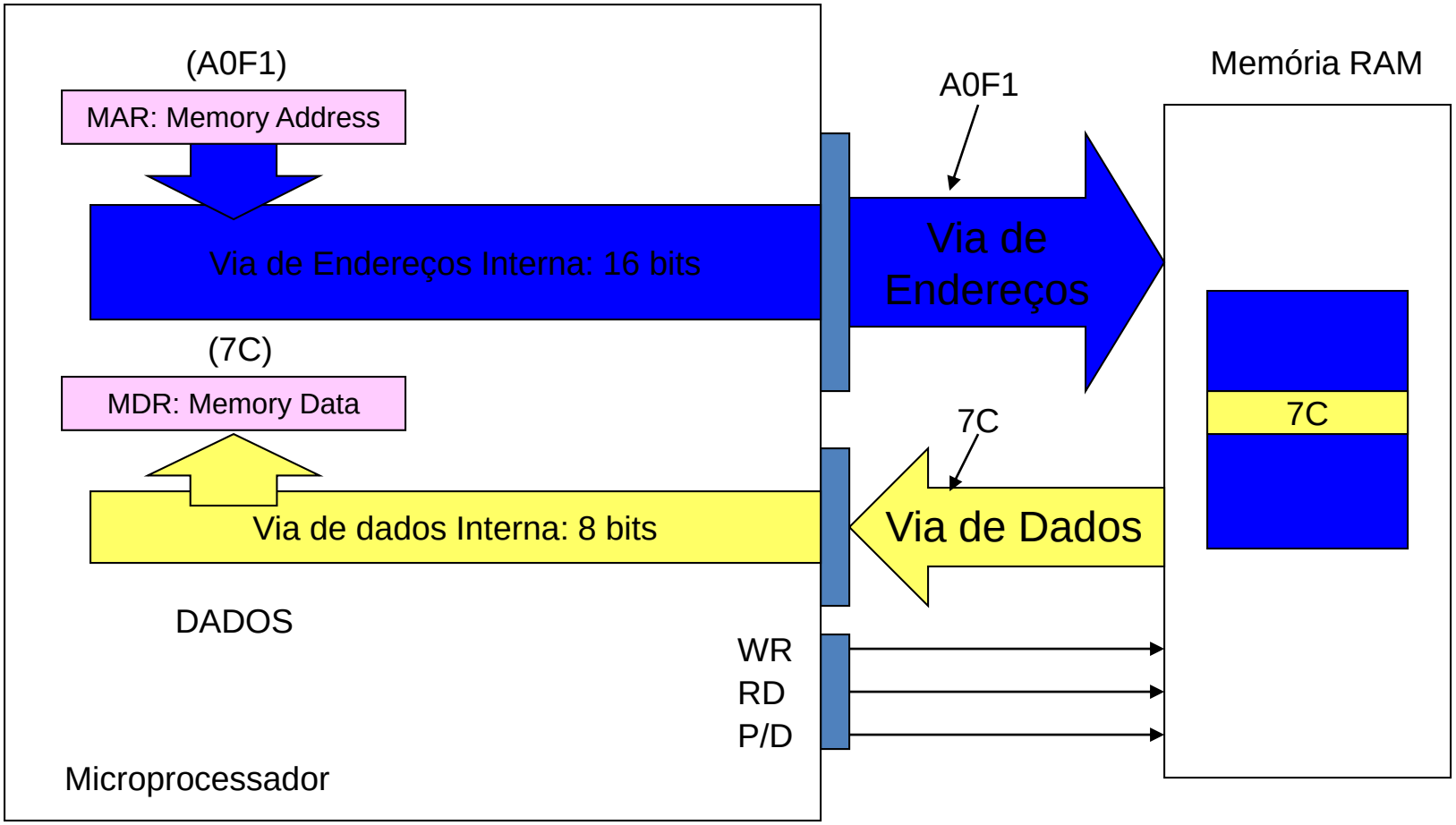
LOAD A0F1: (ACC ← [A0F1])

1: FETCH INSTRUÇÃO
2: PC = PC + 1
3: FETCH ENDEREÇO 1 (+signif.)
4: PC = PC + 1
5: FETCH ENDEREÇO 2 (-signif.)
6: MDR ← [ENDEREÇO]
Q1: MAR → Via de Endereços
Q2: Comando de Leitura
Q3: Memória → Via de Dados
Q4: Via de Dados → MDR

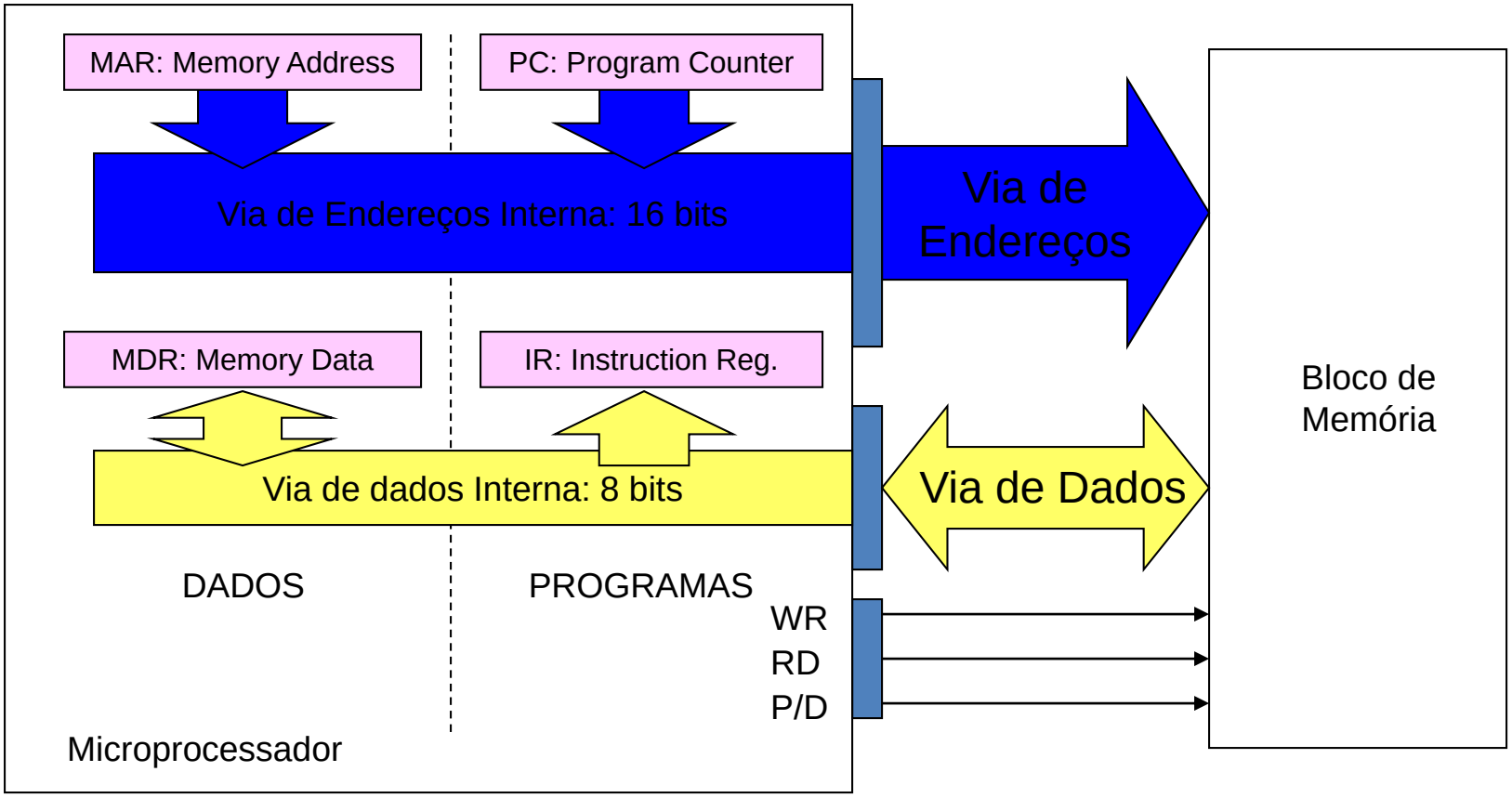
7: Transferência para Acumulador
ACC ← MDR
Q1: ULA: Bloqueia Barramento A
Q2: MDR → Barramento B
Q3: ULA: bypass Barramento B
Q4: Barramento C → Acumulador

8: PC = PC + 1

FLUXO DE INFORMAÇÕES PARA LEITURA



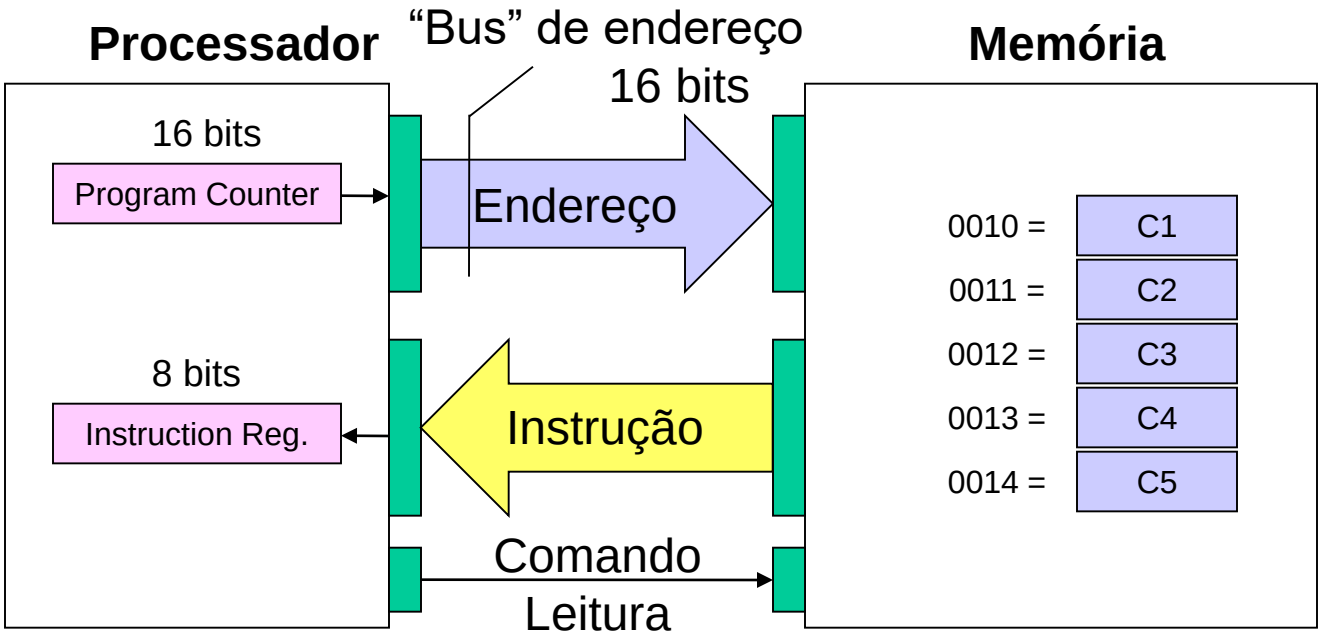
UCP: CONTROLE E ACESSO À MEMÓRIA



Execução da Instrução

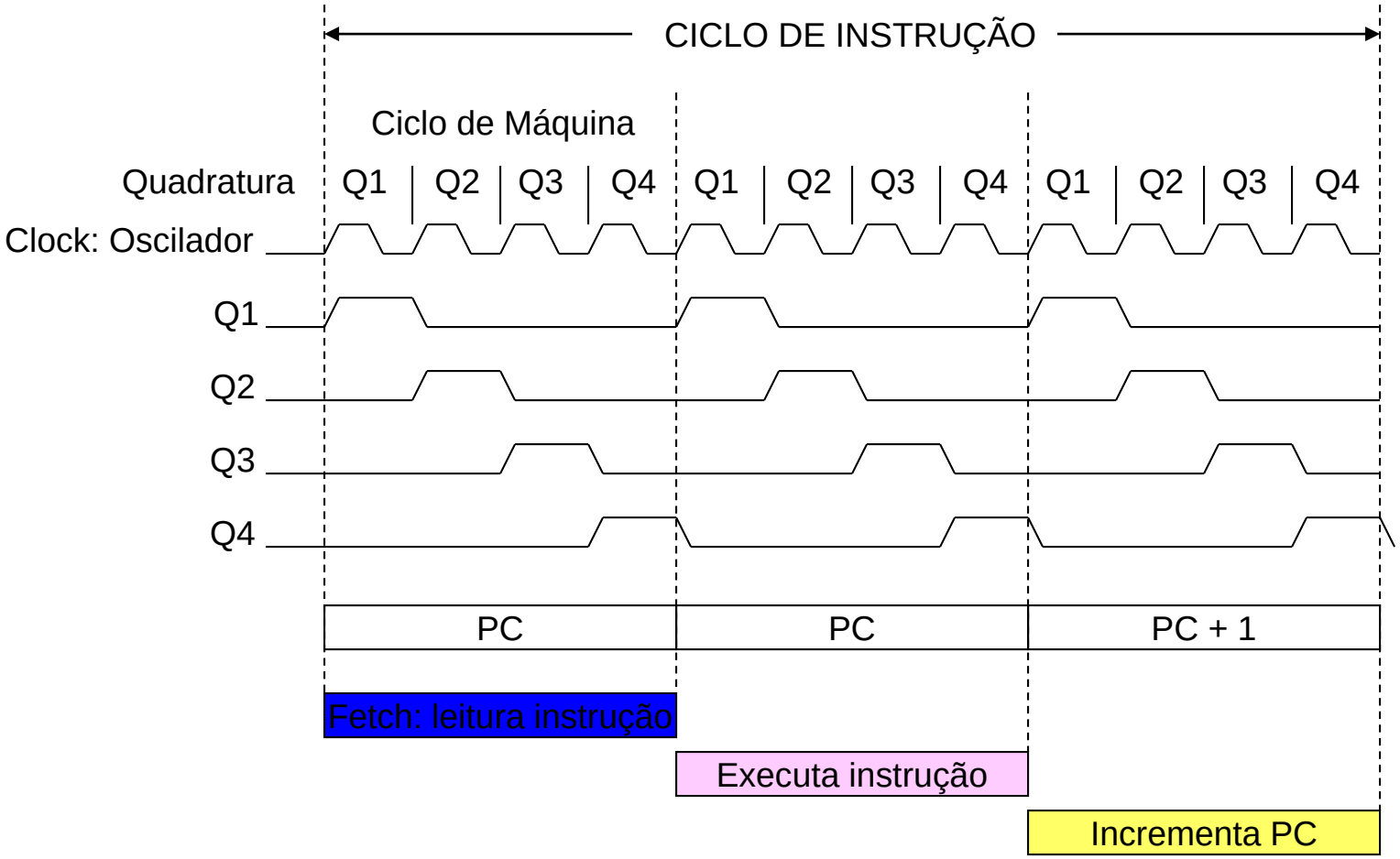
EXECUÇÃO DE UMA INSTRUÇÃO

FETCH: LEITURA DE UMA INSTRUÇÃO DA MEMÓRIA



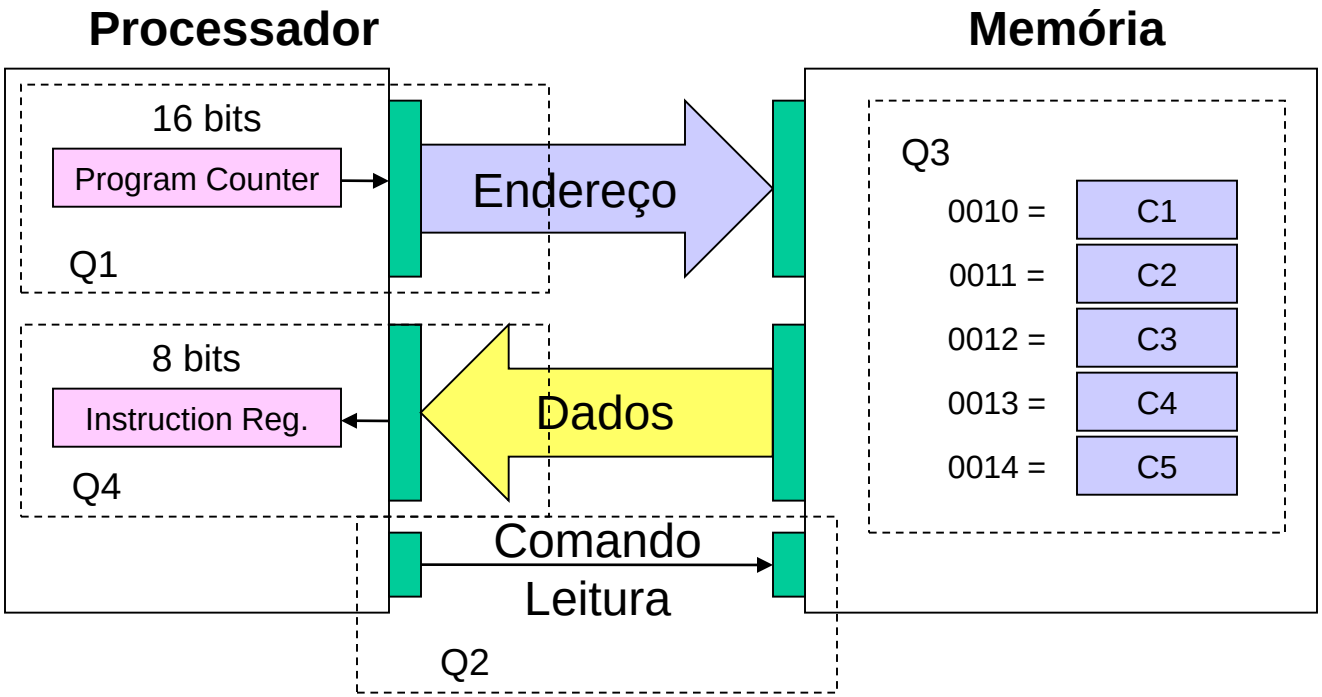
Processador: 8 bits ® dados e instruções 8 bits
Memória: 16 bits de endereço e 8 bits de dados: 64K x 8 = 512K bits

DIAGRAMA DE TEMPOS DO PROCESSADOR



EXECUÇÃO DE UMA INSTRUÇÃO

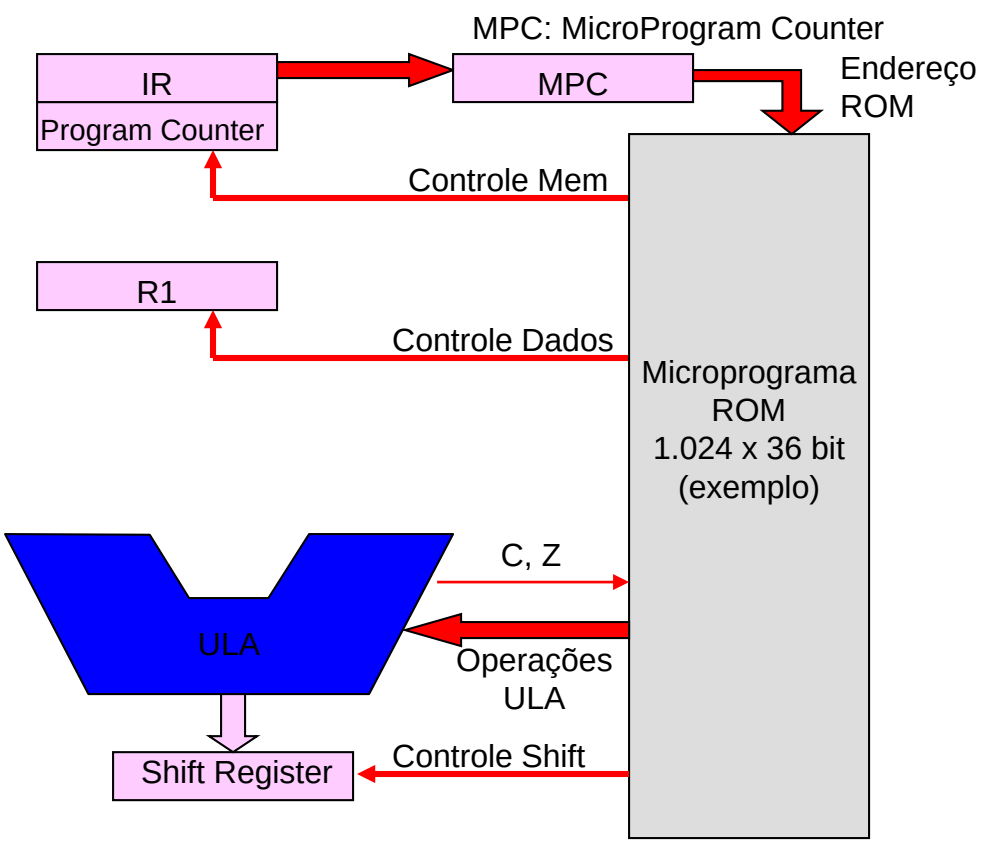
CICLO 1: FETCH



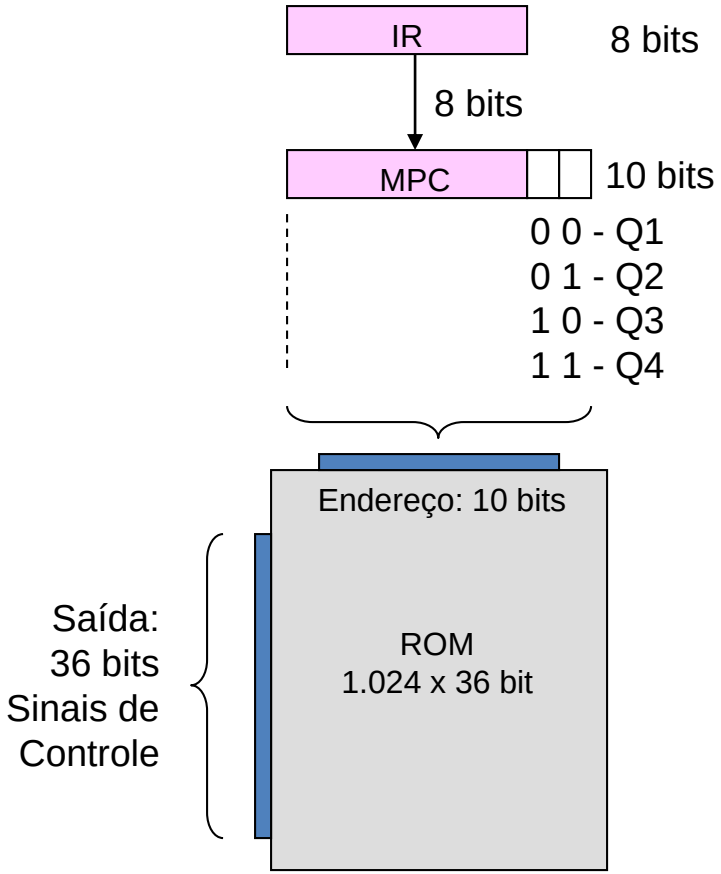
Q1: Processador transfere PC para a Via de Endereços
Q2: Comando de Leitura: Memória decodifica o endereço
Q3: Memória transfere o conteúdo da posição do endereço para Via de Dados
Q4: Processador transfere Bus Dados para IR (registrador de instruções)

EXECUÇÃO DE UMA INSTRUÇÃO

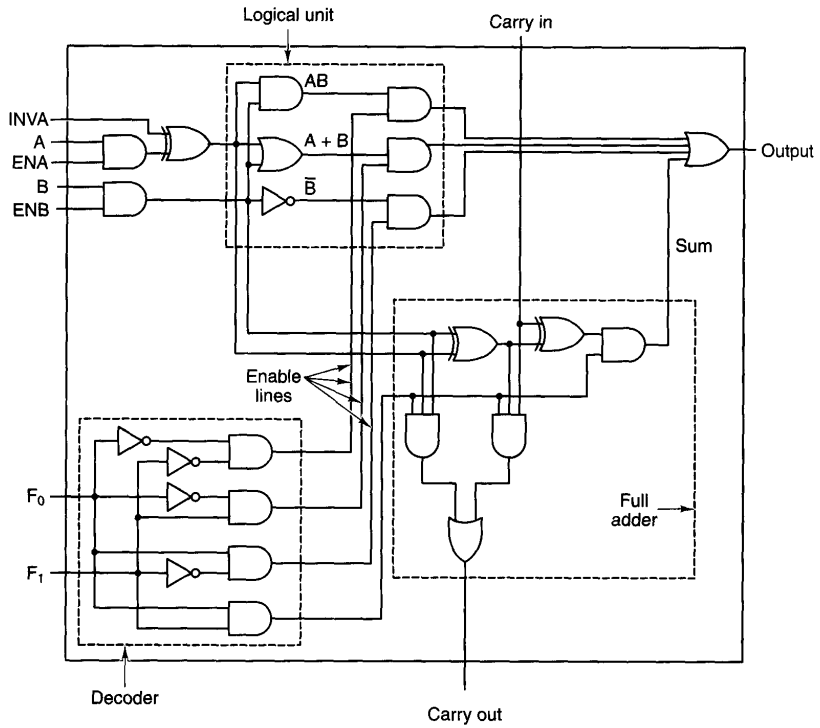
MICROPROGRAMA E SEQUENCIALIZAÇÃO



Exemplo de montagem do MPC



EXECUÇÃO DE UMA INSTRUÇÃO



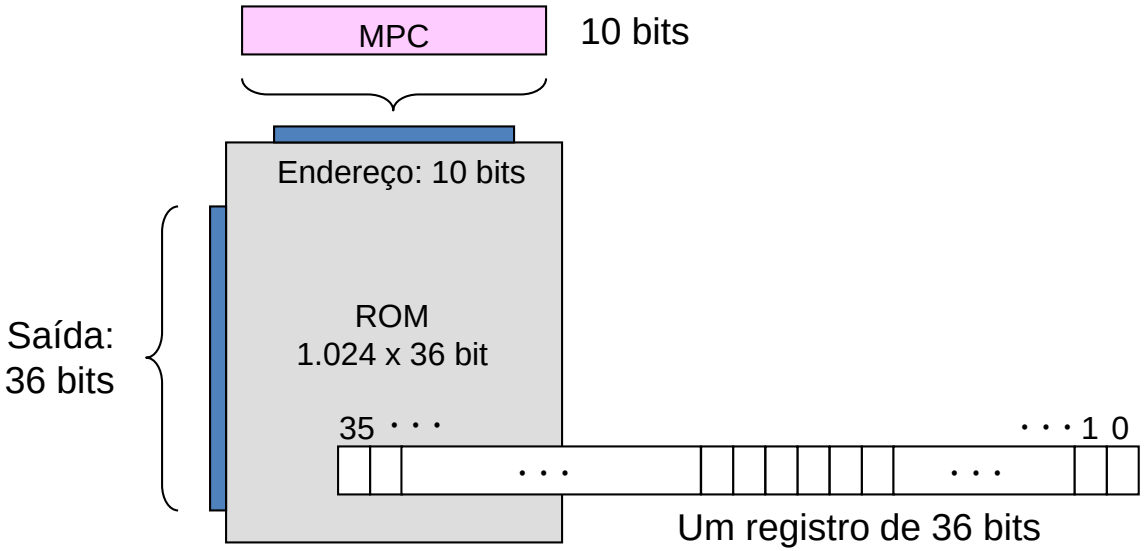
A 1-bit ALU.

EXEMPLOS DE COMANDOS PARA ULA

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	–1

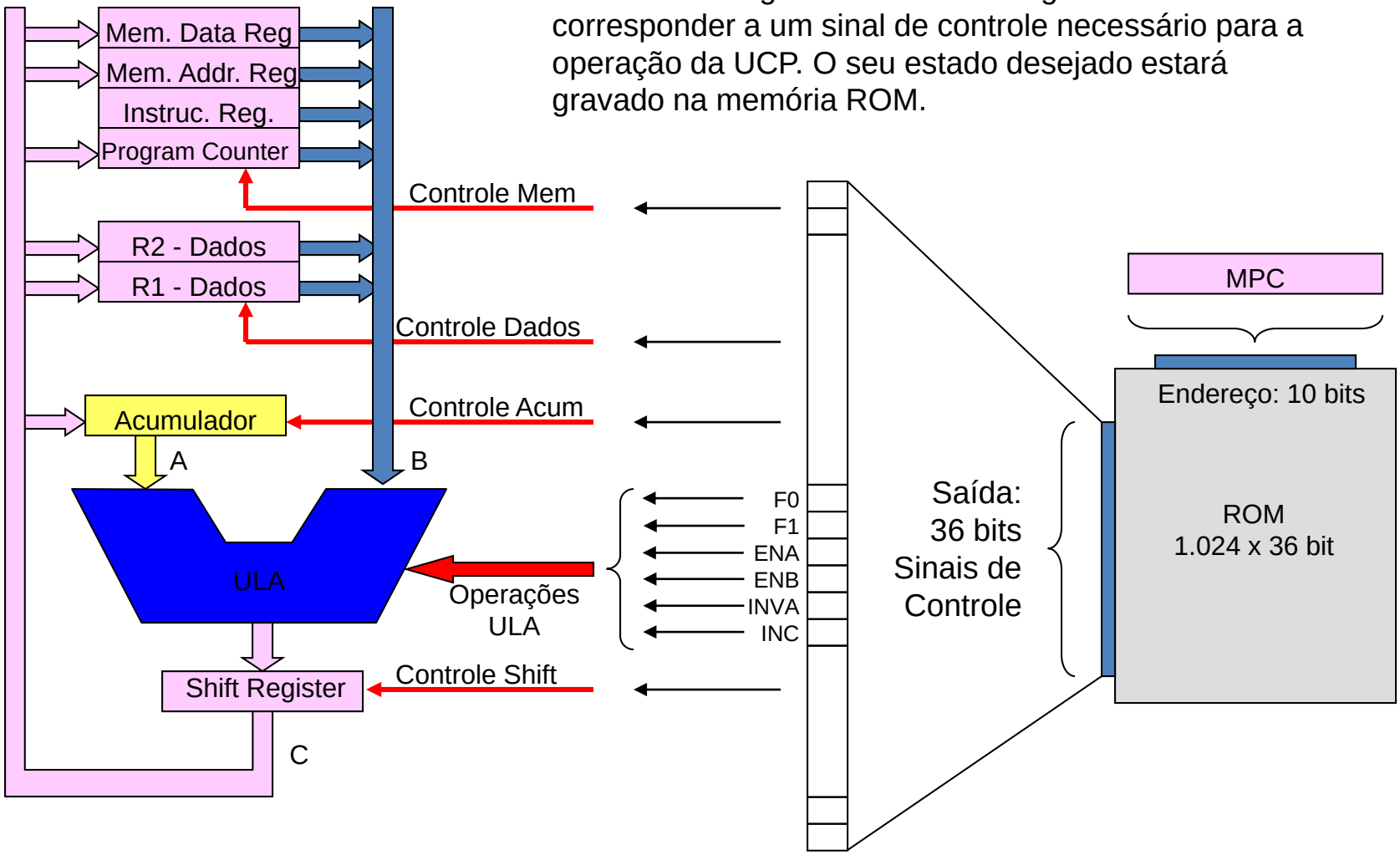
MICROPROGRAMA - 1

- Memória tipo ROM (Read Only Memory), extremamente rápida, com 10 bits de endereço, ou seja 2^{10} ou 1.024 registros.
- Endereço da memória: registrador especial da UCP, chamado de MPC (Micro Program Counter).
- Tamanho do registro: 36 bits.

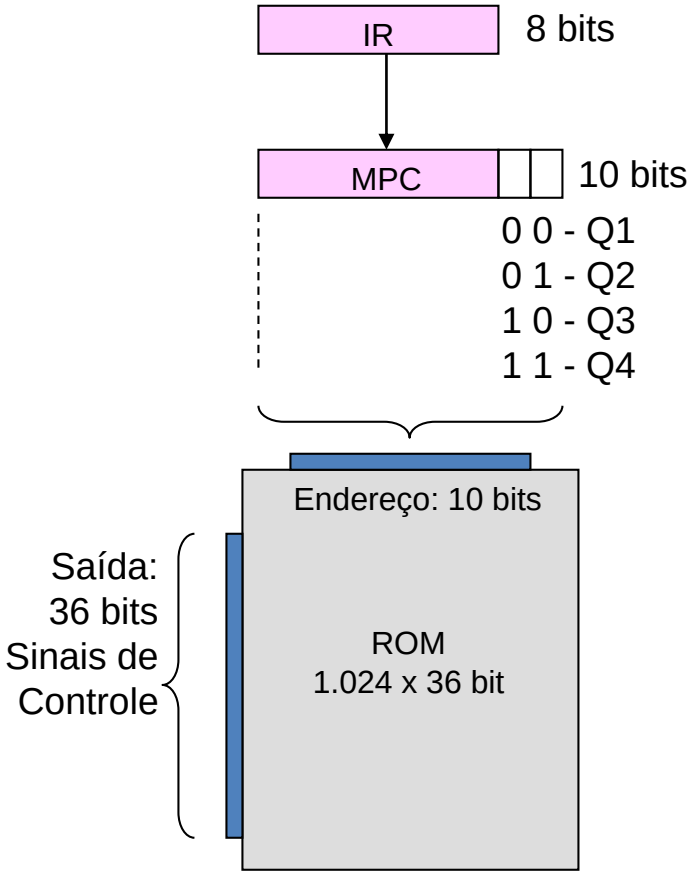


MICROPROGRAMA - 2

Tamanho do registro: cada bit do registro deverá corresponder a um sinal de controle necessário para a operação da UCP. O seu estado desejado estará gravado na memória ROM.



MICROPROGRAMA – 3

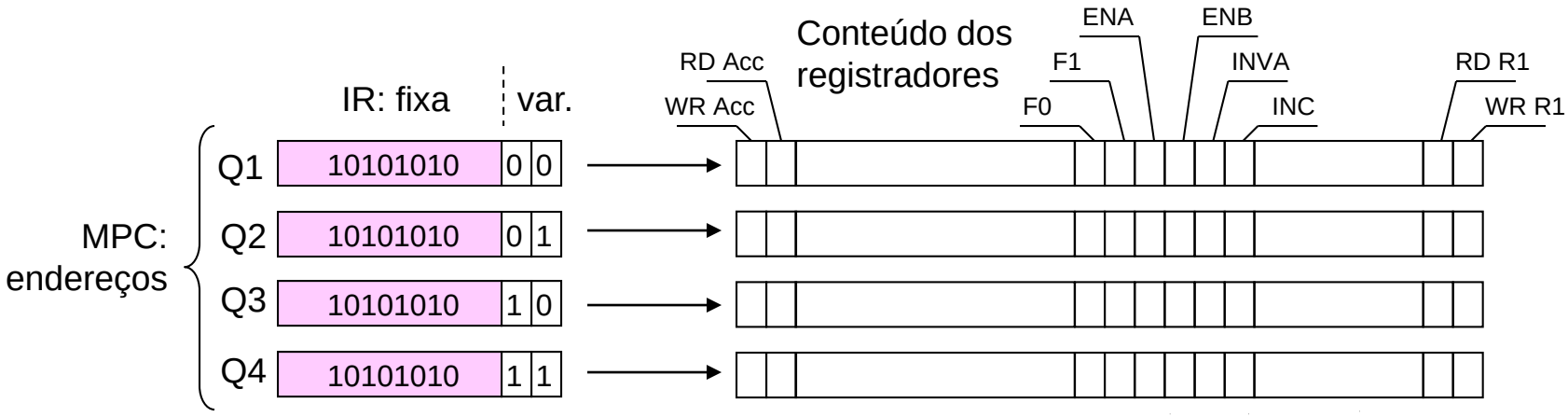


MPC é formado por 10 bits (na nossa UCP didática). Vamos dividir o MPC em duas partes: uma parte fixa formada pelos 8 primeiros bits, e uma parte variável com os dois bits menos significativos. Utilizando um circuito contador de dois bits, e sincronizado pelo relógio da UCP, temos um gerador de endereços do MPC cujos dois bits menos significativos varia de ’00’ a ’11’.

Se a parte fixa de 8 bits for preenchida pelos valores lidos do IR (Instruction Register), temos um endereço de 10 bits que ‘varre’ quatro posições sequenciais, de ‘IR-00’ a ‘IR-11’.

Se para cada uma das quatro posições, atribuirmos ao conteúdo do registrador os sinais necessários para a realização das funções correspondentes aos quatro passos Q1, Q2, Q3 e Q4, implementaremos o mecanismo necessário para a execução do **Ciclo de Máquina da Instrução**.

MICROPROGRAMA – 4



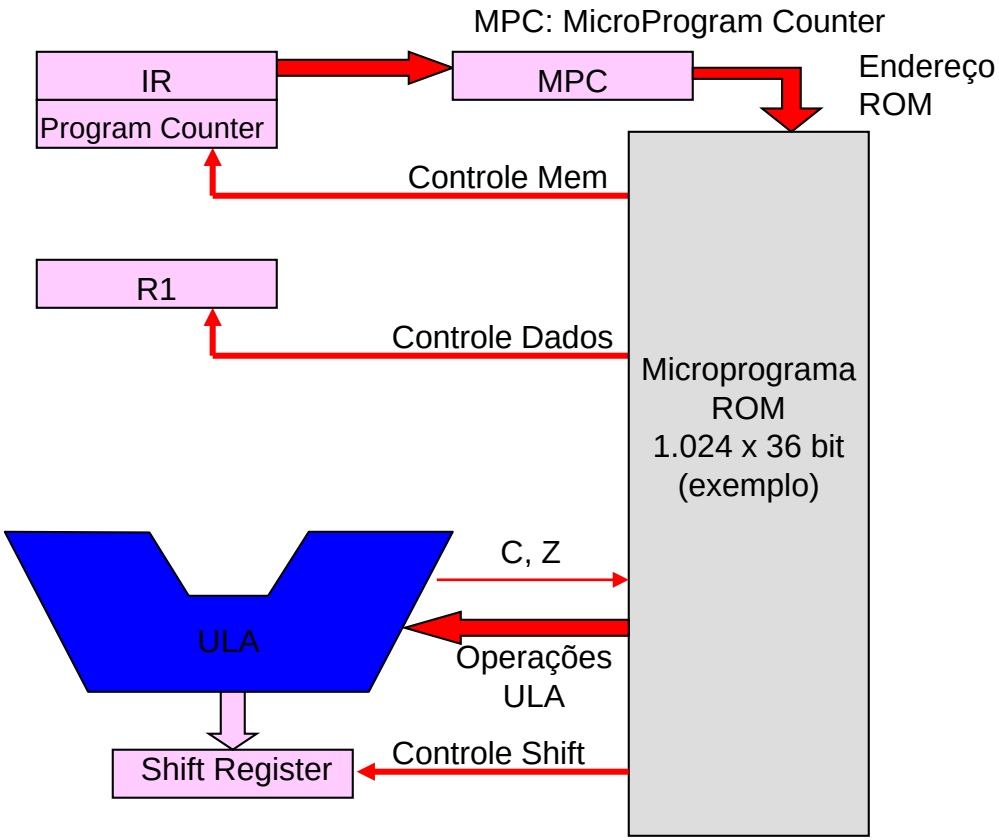
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
0	1	0	0	0	1	1
0	1	0	0	1	0	–1

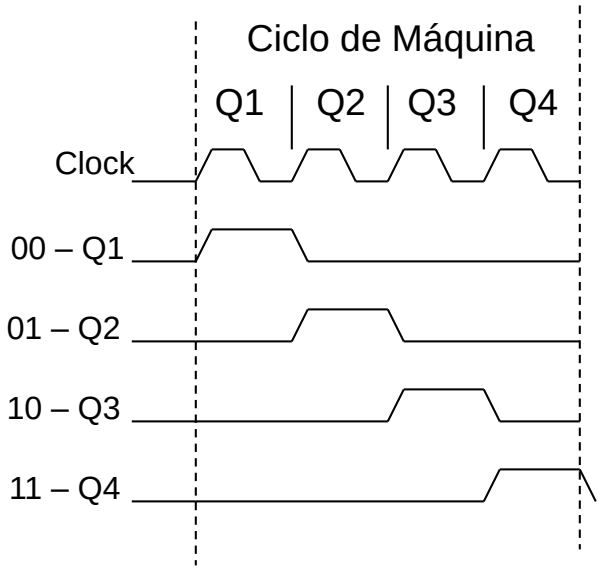
Exemplo:
Acumulador = Acumulador AND R1

Execução:
Q1: R1 → Barramento B
Q2: ULA: Operação AND
Q3: ULA: Retenção Barramento A
Q4: Barramento C → Acumulador

MICROPROGRAMA – 5

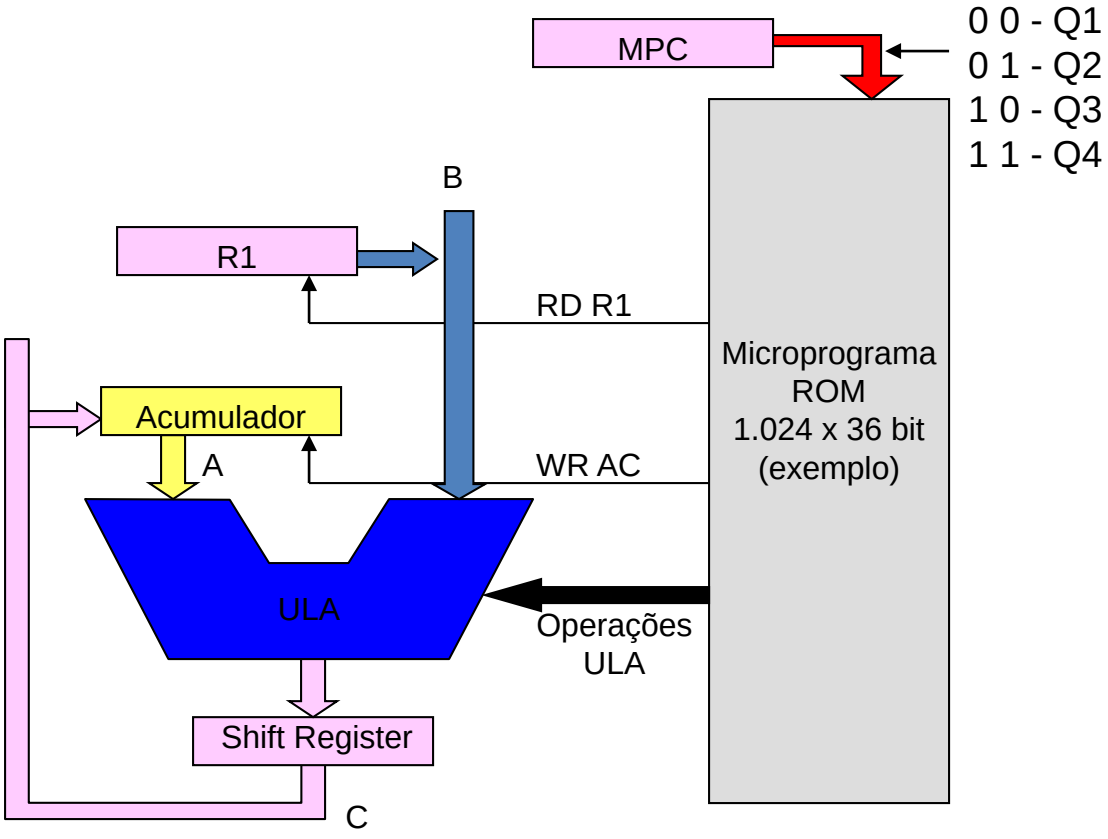
RESUMO





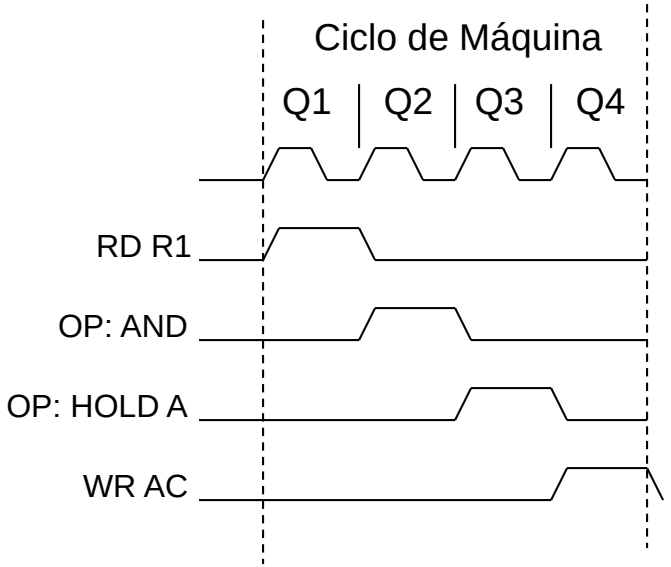
EXECUÇÃO DE UMA INSTRUÇÃO

CICLO 2: EXECUÇÃO DA INSTRUÇÃO:



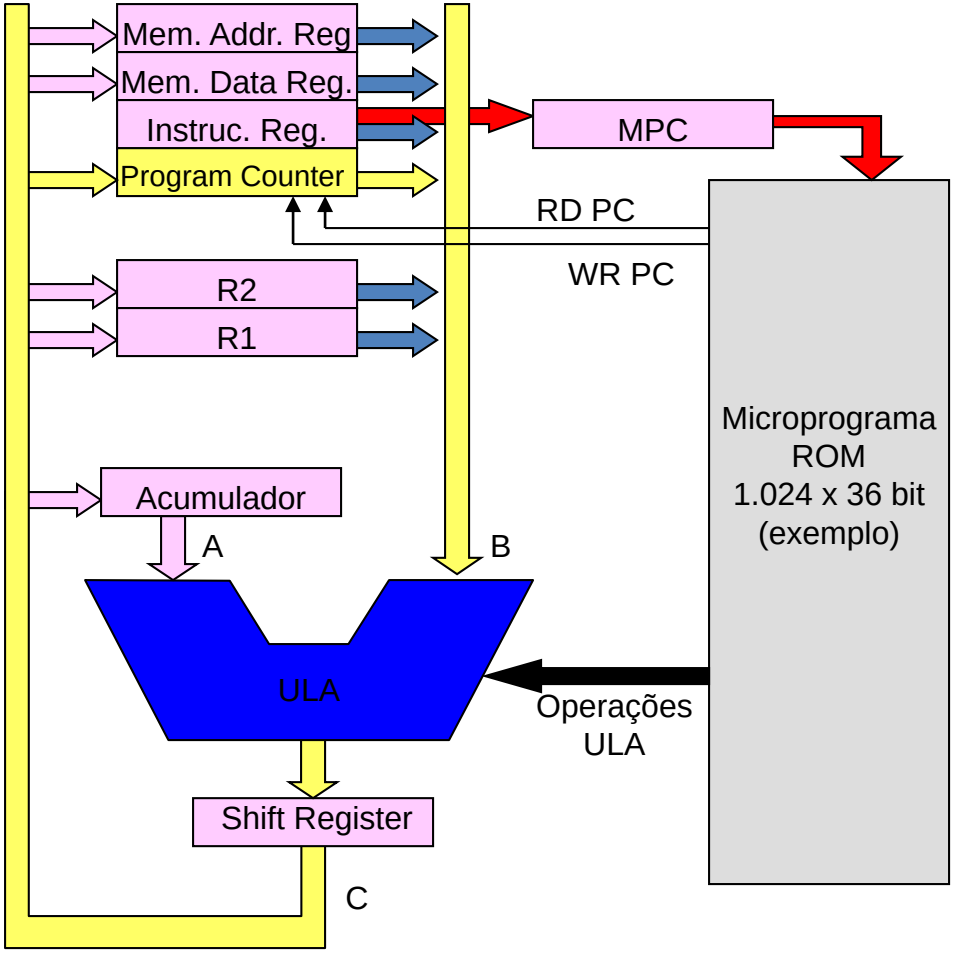
Exemplo:
Acumulador = Acumulador AND R1

Execução:
Q1: R1 → Barramento B
Q2: ULA: Operação AND
Q3: ULA: Retenção Barramento A
Q4: Barramento C → Acumulador



EXECUÇÃO DE UMA INSTRUÇÃO

CICLO 3: INCREMENTA O PROGRAM COUNTER (PC):



PC = PC + 1

Execução:
Q1: ULA: Bloqueia Barramento A
Q2: PC → Barramento B
Q3: ULA: Incrementa Barramento B
Q4: Barramento C → PC

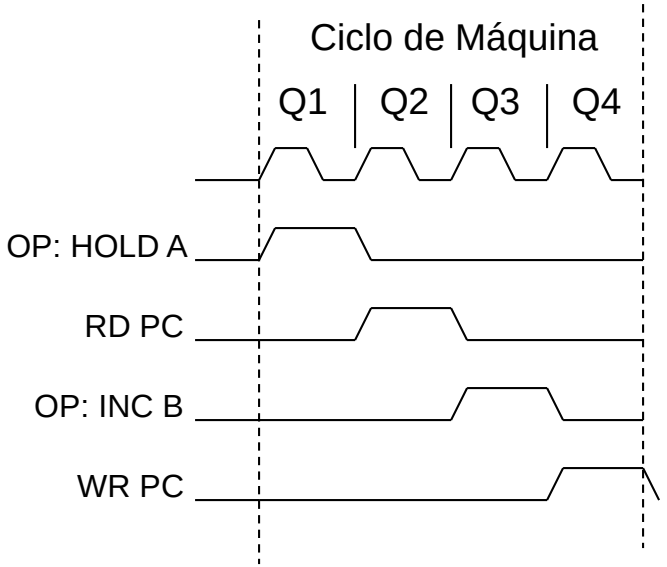
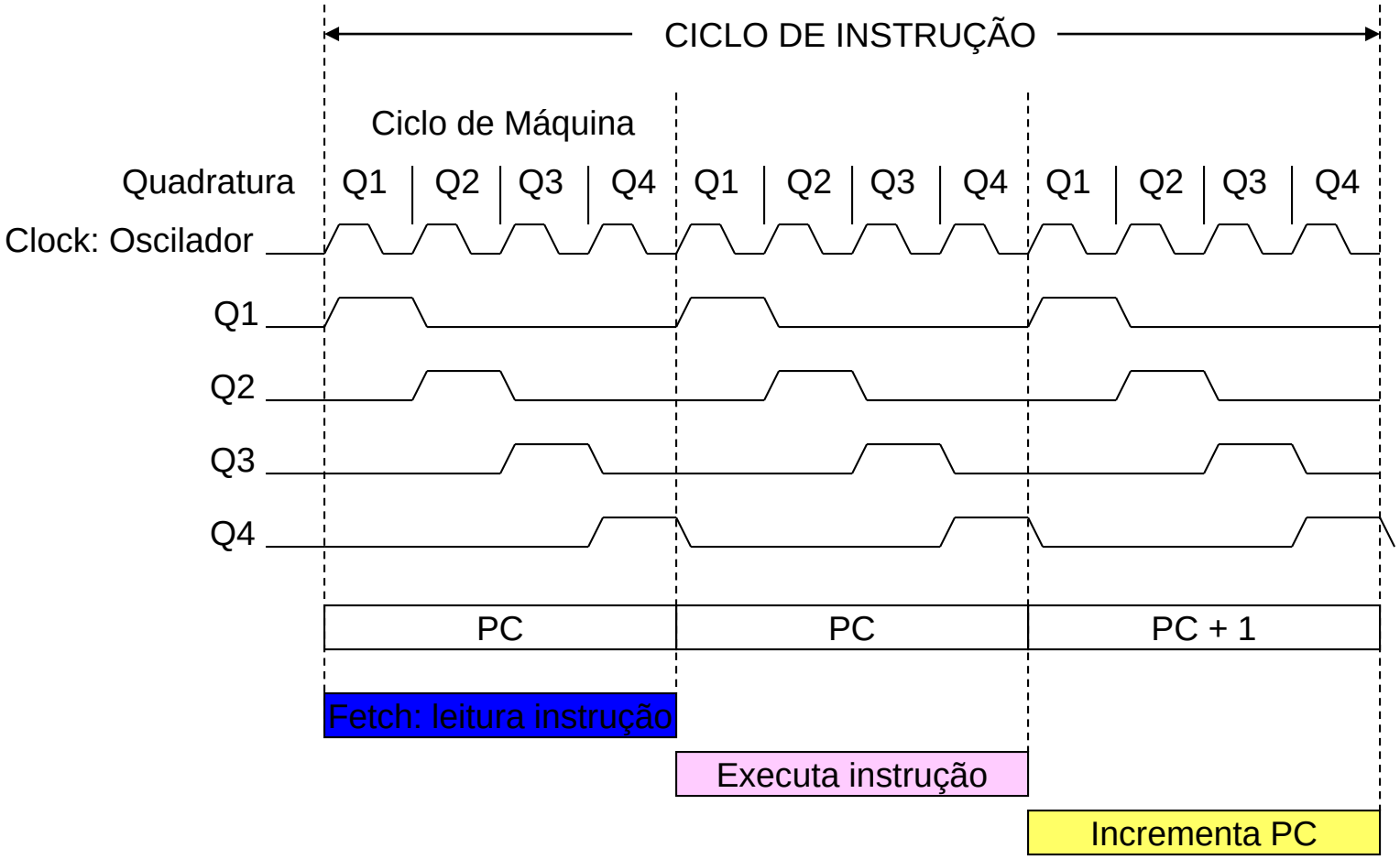
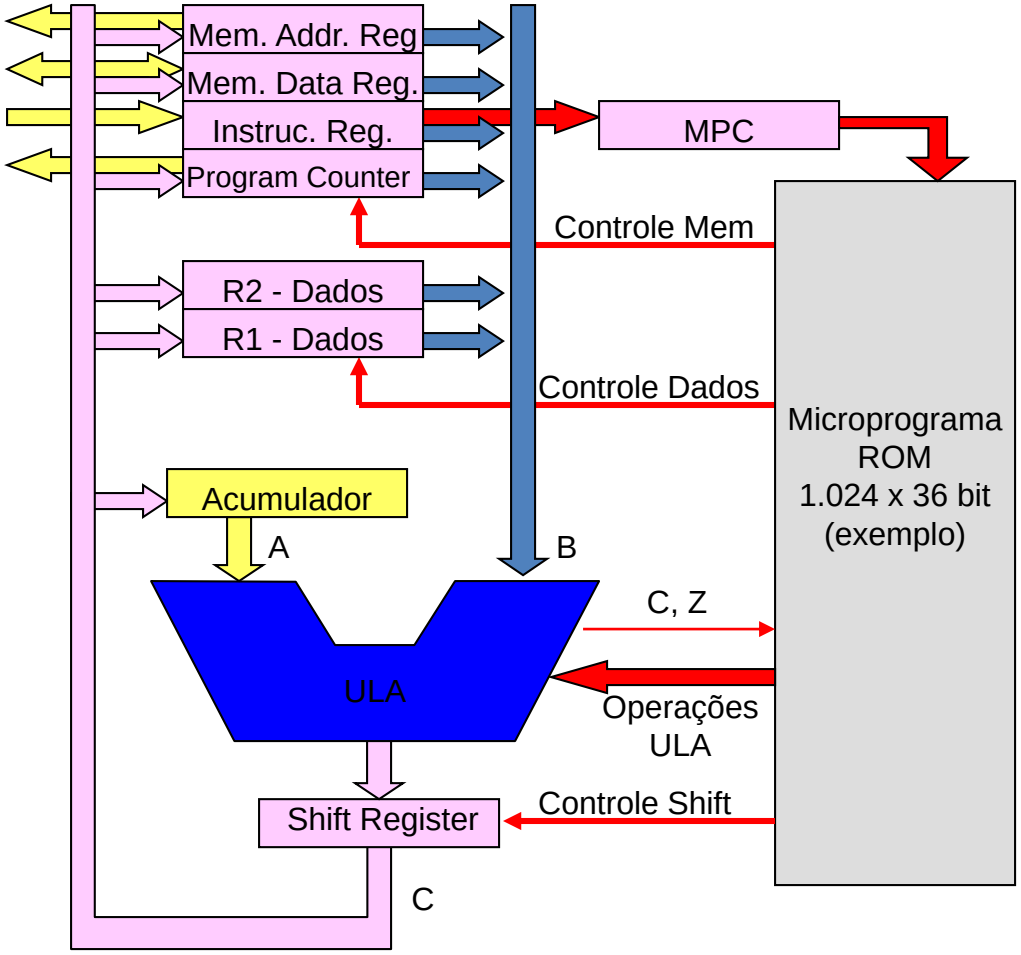


DIAGRAMA DE TEMPOS DO PROCESSADOR



EXECUÇÃO DE UMA INSTRUÇÃO

RESUMO: EXECUÇÃO DA INSTRUÇÃO ACC ← ACC *AND* R1



Um ciclo de Instrução =
Três ciclos de máquina

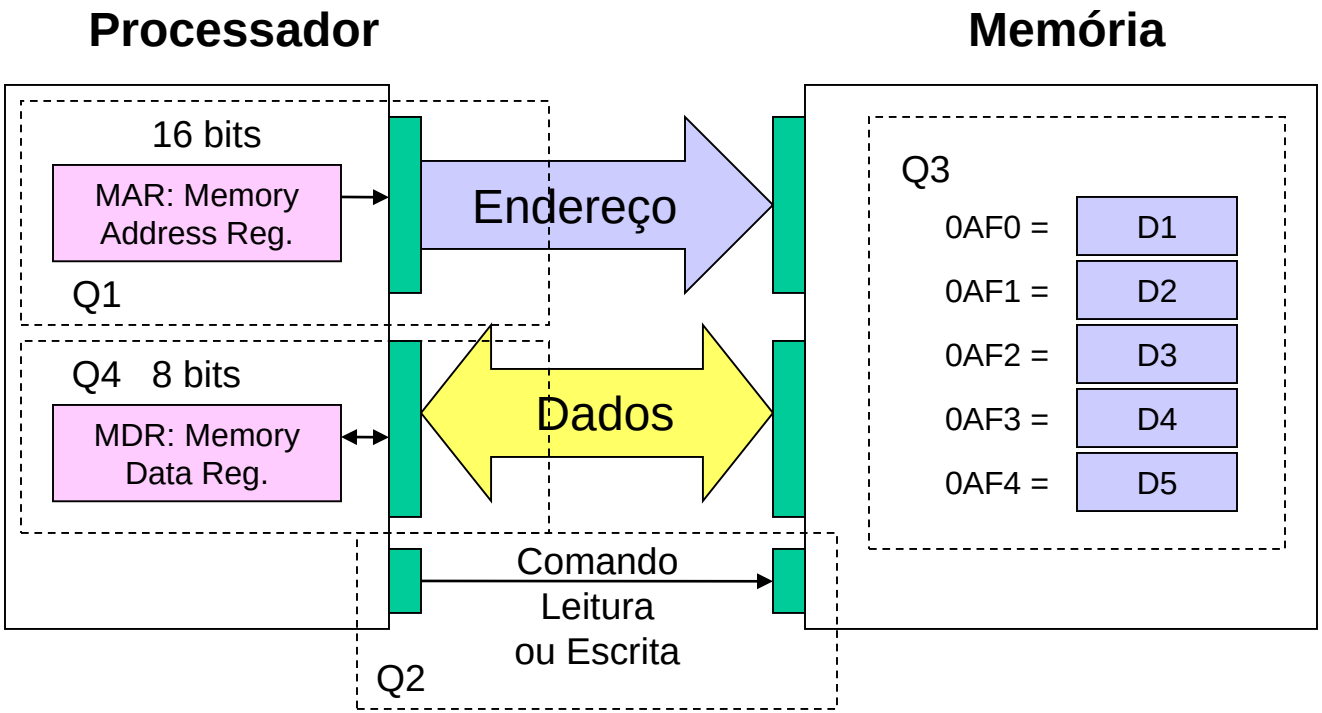
1: FETCH
Q1: PC → Via de Endereços
Q2: Comando de Leitura
Q3: Memória → Via de Dados
Q4: Via de Dados → IR

2: Acumulador = Acumulador AND R1
Q1: R1 → Barramento B
Q2: ULA: Operação AND
Q3: ULA: Retenção Barramento A
Q4: Barramento C → Acumulador

3: PC = PC + 1
Q1: ULA: Bloqueia Barramento A
Q2: PC → Barramento B
Q3: ULA: Incrementa Barramento B
Q4: Barramento C → PC

Mapeamento dos Endereços

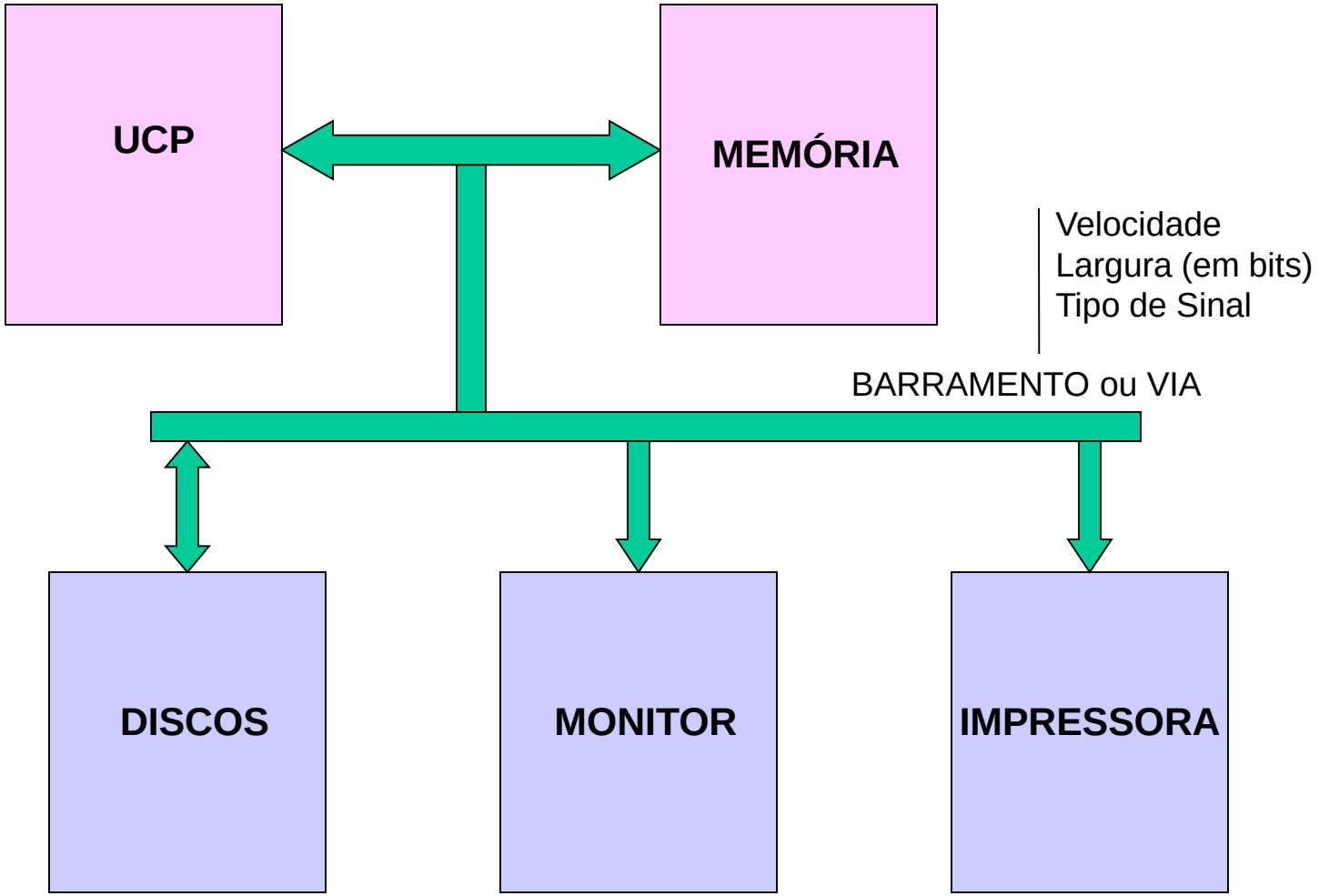
ACESSO À MEMÓRIA DE DADOS: CICLO DE INSTRUÇÕES



LEITURA DA MEMÓRIA
Q1: MAR → Via de Endereços
Q2: Comando de Leitura: Memória decodifica
Q3: Memória → Via de Dados
Q4: Via de Dados → MDR

ESCRITA NA MEMÓRIA
Q1: MAR → Via de Endereços
MDR → Via de Dados
Q2: Comando de Escrita: Memória decodifica
Q3: Via de Dados → Memória
Q4: Estabilização Memória

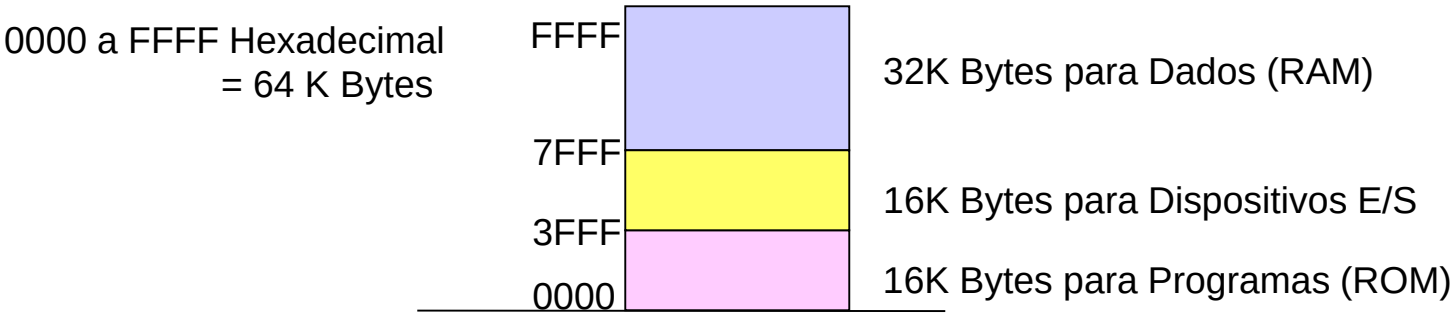
UCP E ACESSO A OUTROS DISPOSITIVOS



ENDEREÇAMENTO DE DISPOSITIVOS E/S

EXEMPLO PARA DISPOSITIVOS MAPEADOS EM MEMÓRIA

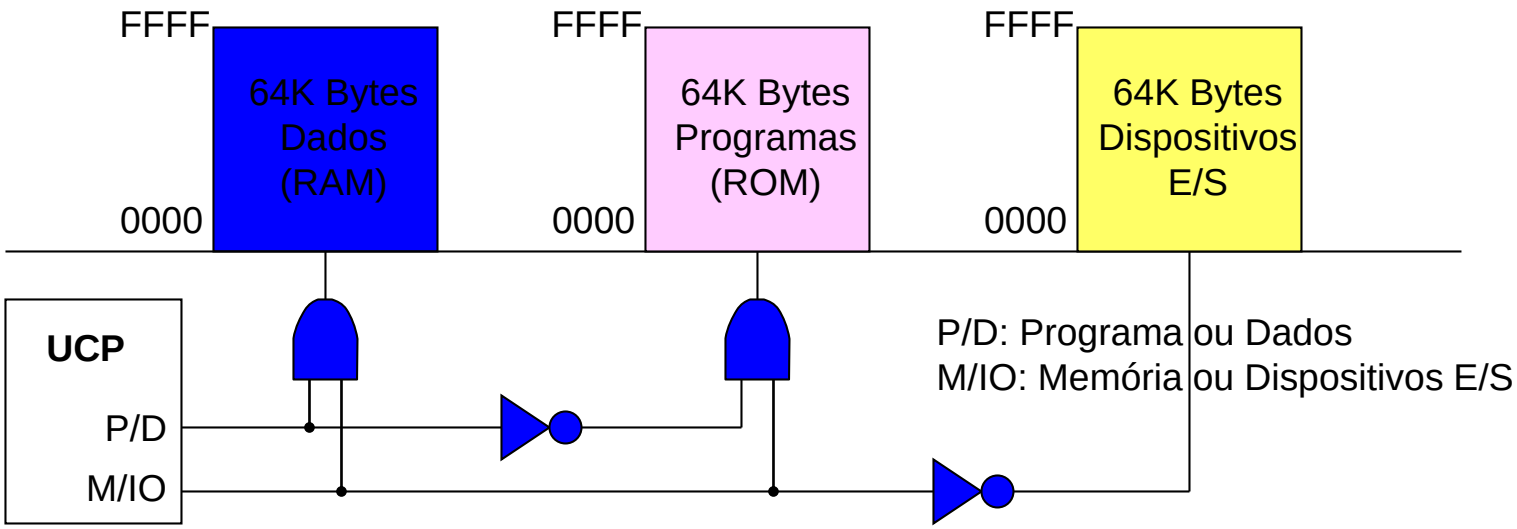
Área Total de 64K Bytes: Divididos com Programas, Dados e Dispositivos E/S



ENDEREÇAMENTO DE DISPOSITIVOS E/S

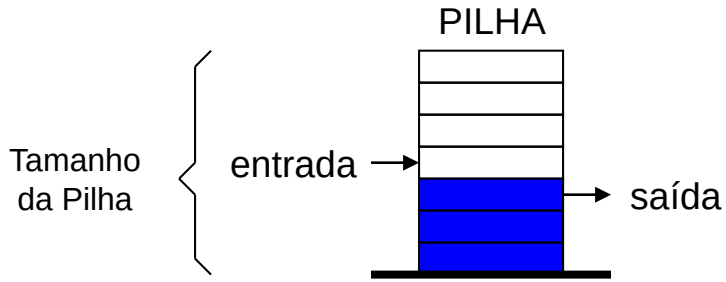
EXEMPLO PARA ÁREA DE MEMÓRIA DE 64K Bytes:

Áreas Separadas de 64K Bytes para Programas, Dados e Dispositivos E/S (uso dos sinais M/IO e P/D)



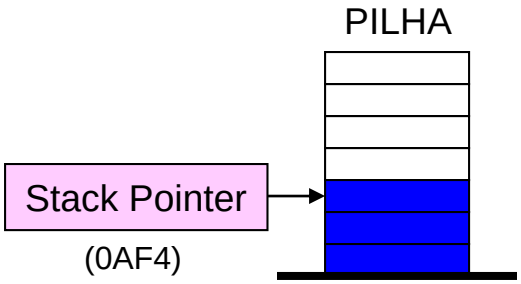
Pilhas e Stack Pointer

STACK-POINTER: PONTEIRO DE PILHA

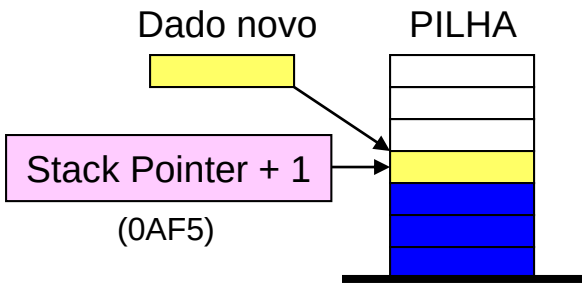


Pilha:
Estrutura de Dados do tipo FILO
(First In Last Out)

Stack Pointer: contém o endereço da memória (ponteiro) onde está localizado o último dado escrito (inserido) na pilha.
Exemplo: Se o último dado da pilha estiver gravado no endereço 0AF4 (em hexadecimal), o ponteiro irá conter o valor 0AF4.

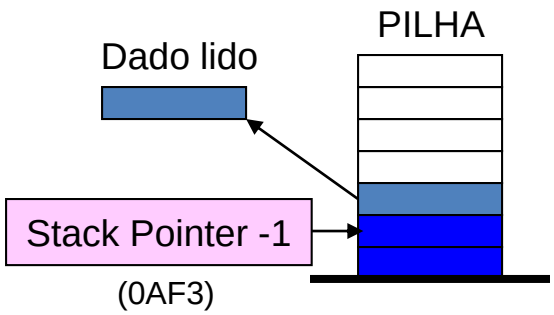


INSERIR NA PILHA: *PUSH*



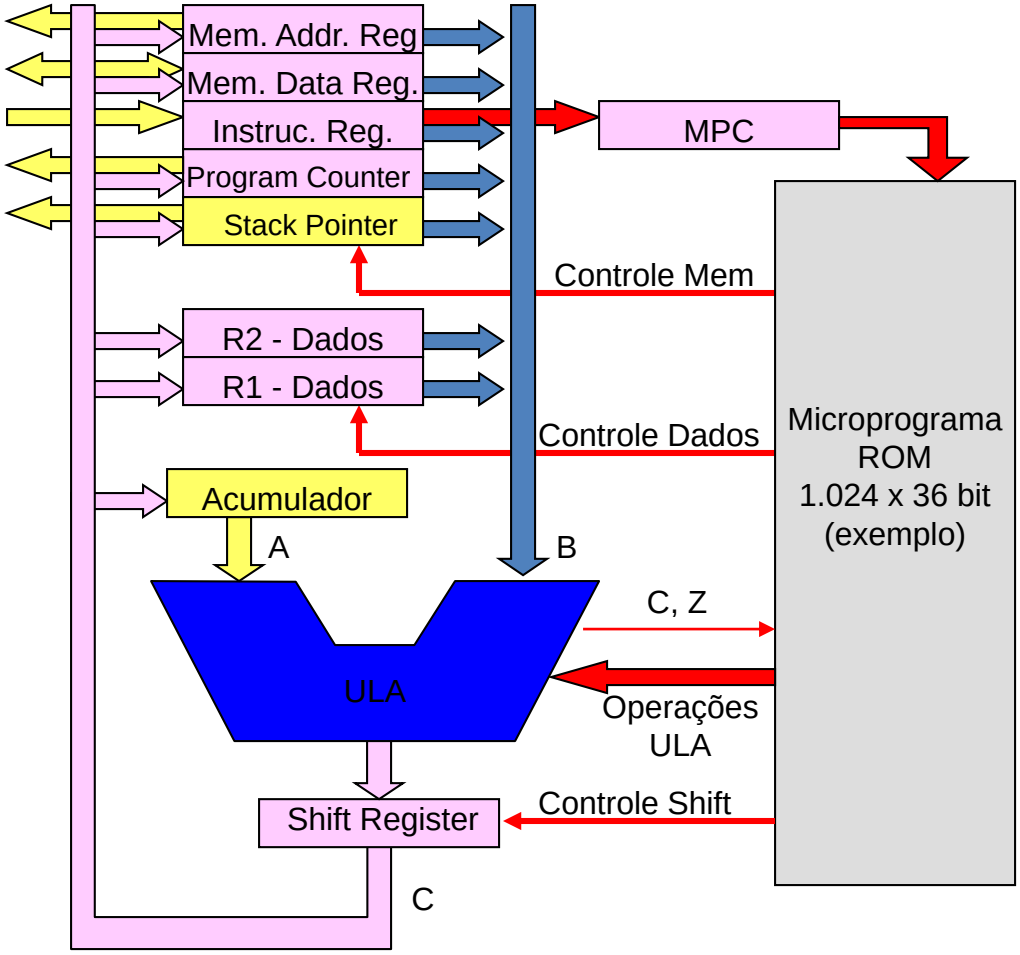
Incrementa SP **antes**

RETIRAR DA PILHA: *POP*



Decrementa SP **depois**

INSTRUÇÃO INSERÇÃO NA PILHA: PUSH ACC



Quatro ciclos de máquina

1: FETCH da Instrução PUSH
Q1: PC → Via de Endereços
Q2: Comando de Leitura
Q3: Memória → Via de Dados
Q4: Via de Dados → IR

2: SP = SP + 1
Q1: ULA: Bloqueia Barramento A
Q2: SP → Barramento B
Q3: ULA: Incrementa Barramento B
Q4: Barramento C → SP

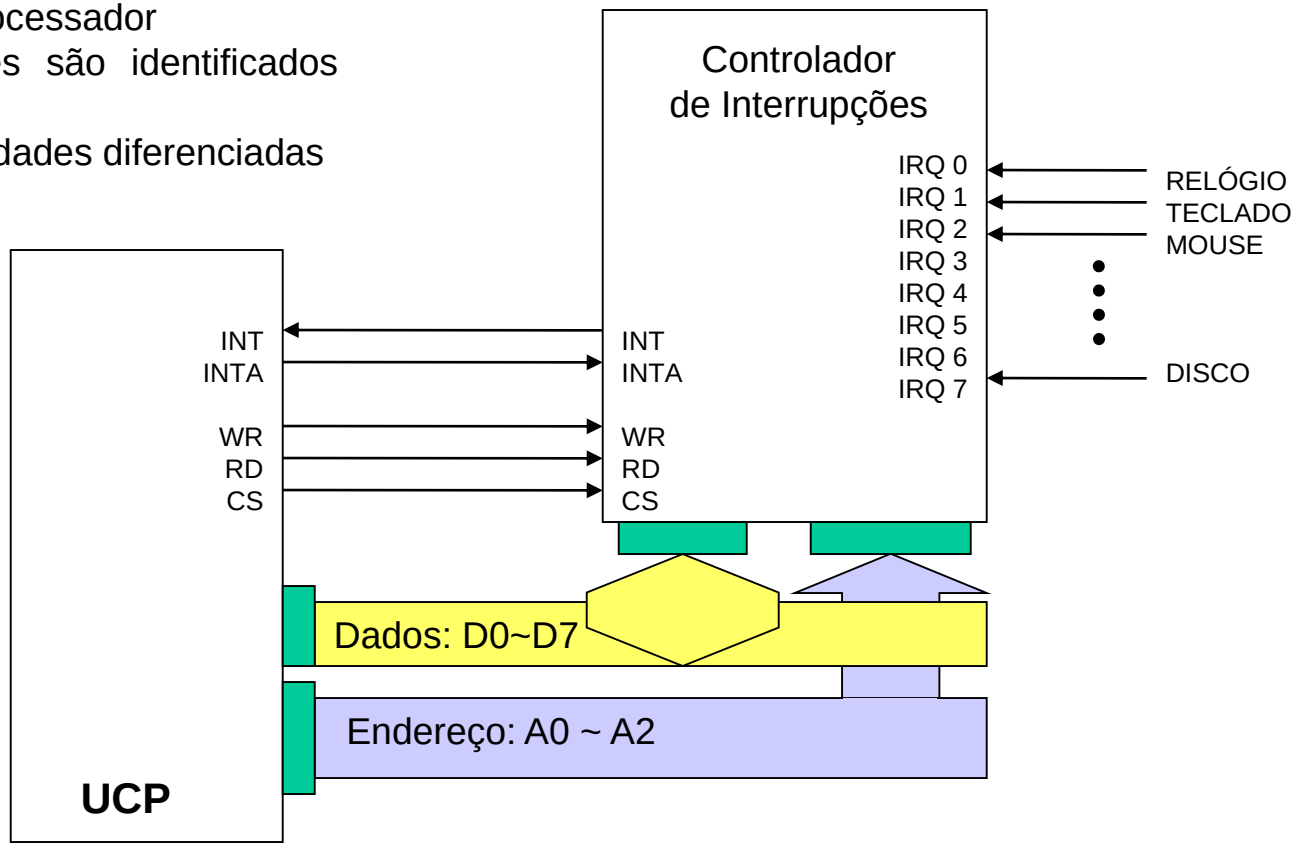
3: ESCRITA NA MEMÓRIA
Q1: SP → Via de Endereços
ACC → Via de Dados
Q2: Comando de Escrita
Q3: Via de Dados → Memória
Q4: Estabilização Memória

4: PC = PC + 1
Q1: ULA: Bloqueia Barramento A
Q2: PC → Barramento B
Q3: ULA: Incrementa Barramento B
Q4: Barramento C → PC

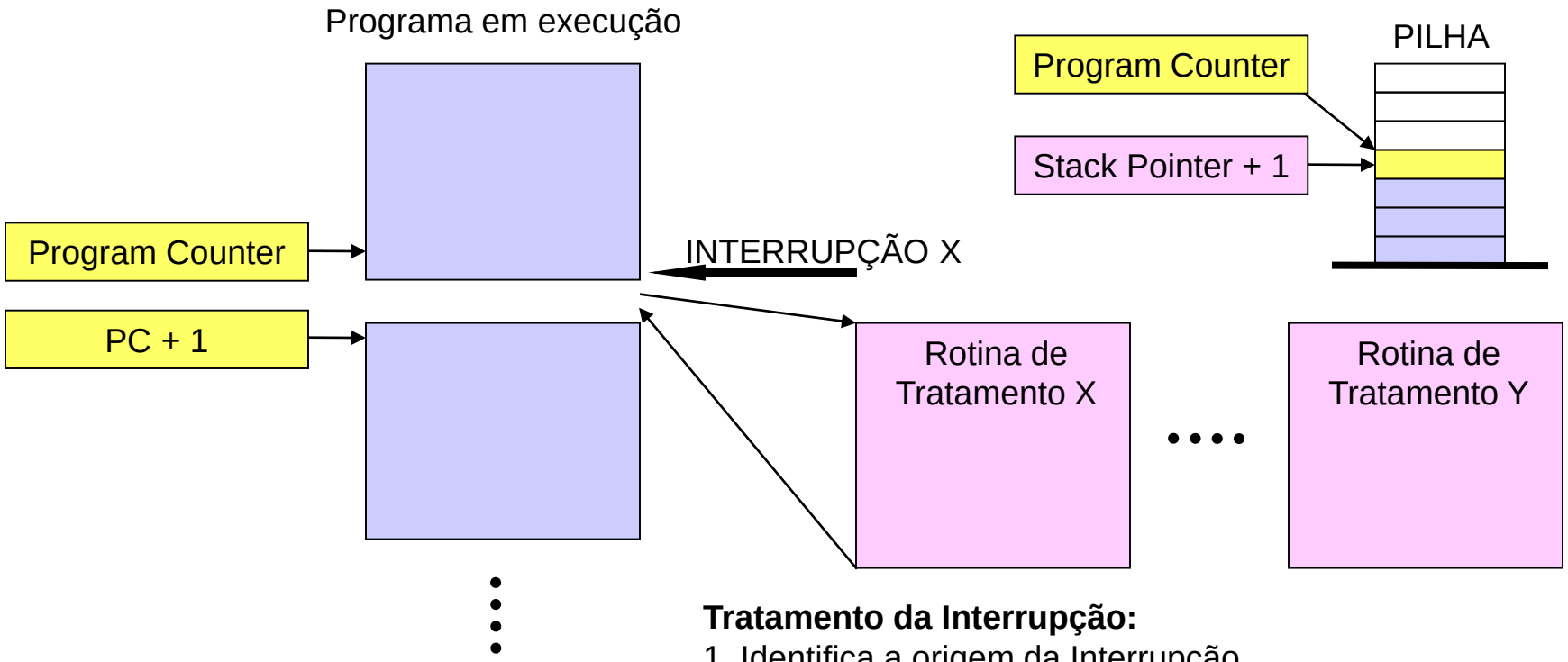
Interrupção

INTERRUPÇÕES

SOLICITAÇÃO DE INTERRUPÇÃO
- Enviados pelos dispositivos que necessitam de um atendimento imediato pelo processador
- Interrupções são identificados individualmente
- Possuem prioridades diferenciadas



SOFTWARE: TRATAMENTO DE INTERRUPÇÕES



Tratamento da Interrupção:

1. Identifica a origem da Interrupção
2. Salva o Program Counter (PC) na pilha
3. Carrega o PC com o endereço da rotina de tratamento correspondente à Interrupção
4. Executa a rotina de tratamento
5. Retorna ao ponto antes da Interrupção: (retira o valor do PC da pilha e restaura)

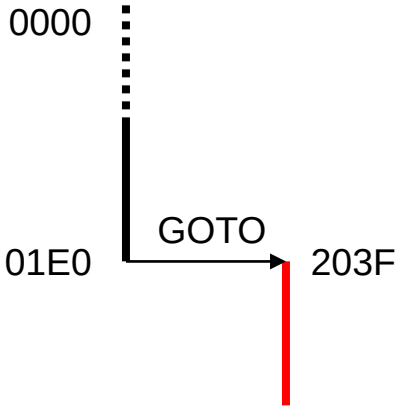
INSTRUÇÕES DE TRÊS BYTES - 2

Exemplo:
GOTO 203F

SALTO INCONDICIONAL: INSTRUÇÃO GOTO ou JUMP

Especificar o endereço de 16 bits da próxima posição do programa

1: FETCH INSTRUÇÃO
2: PC = PC + 1
3: FETCH ENDEREÇO 1 (+signif.)
4: PC = PC + 1
5: FETCH ENDEREÇO 2 (-signif.)
6: PC ← ENDEREÇO

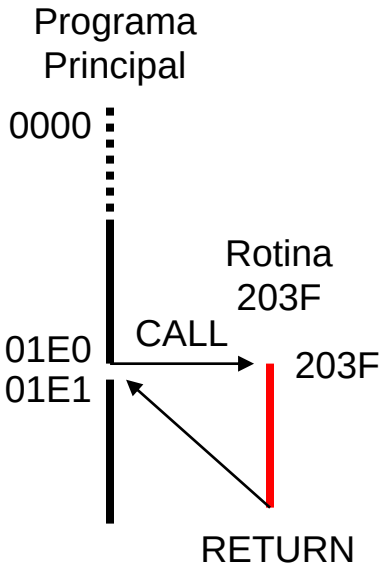


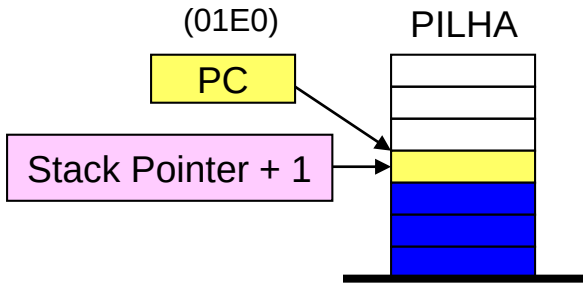
	Program Counter
Antes da instrução GOTO:	01E0
Depois da instrução:	203F

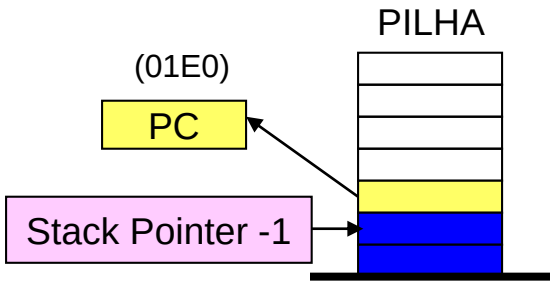
INSTRUÇÕES DE TRÊS BYTES - 3

CHAMADA DE ROTINA: INSTRUÇÃO CALL

Especificar o endereço de 16 bits da posição da rotina







Exemplo:

CALL 203F

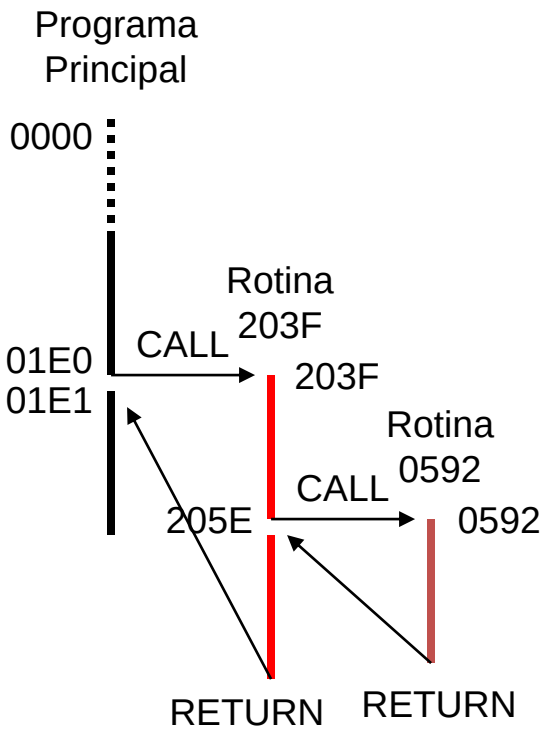
1: FETCH INSTRUÇÃO
2: PC = PC + 1
3: FETCH ENDEREÇO 1 (+signif.)
4: PC = PC + 1
5: FETCH ENDEREÇO 2 (-signif.)
6: SP = SP + 1
7: [SP] ← PC
8: PC ← ENDEREÇO

RETURN

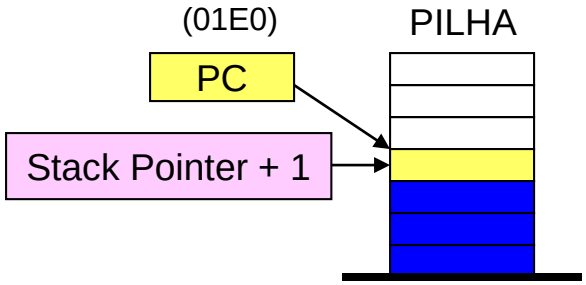
1: FETCH INSTRUÇÃO
2: PC ← [SP]
3: SP = SP - 1
4: PC = PC + 1

NÍVEIS DE SUB-ROTINAS

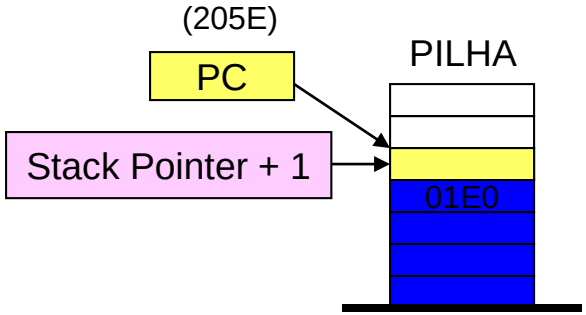
Chamada de Rotinas de dentro das Rotinas



CALL 203F



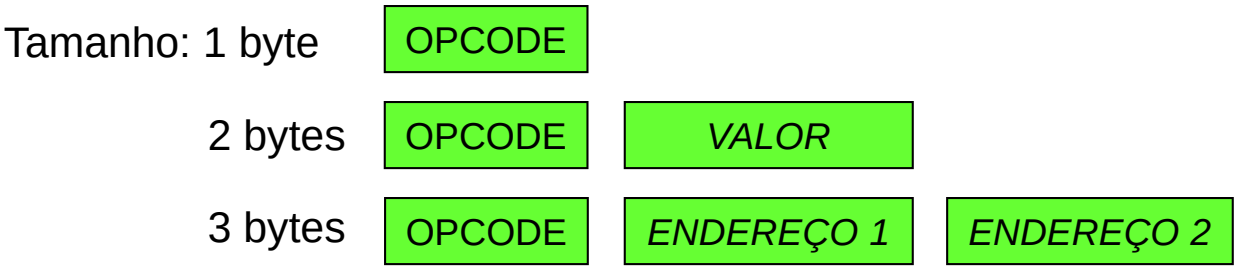
CALL 0592



Estouro da Pilha: Stack Pointer Overflow

INSTRUÇÕES: RESUMO

- Instruções são executadas em ciclos de máquinas, que estabelecem os passos necessários até a sua execução. O número de ciclos depende da instrução.
- Instruções possuem tamanhos diferentes em bytes, sendo compostas por um código de operação (Op Codes) seguido por parâmetros que podem ser valores constantes, endereços, etc.



- Etapas típicas de um ciclo de instruções:
  1 - Fetch da instrução, da memória para o registrador de instruções (IR)
  2 - Incrementa o contador de instruções (PC), apontando-o para a próxima instrução
  3 - Determina o tipo de instrução carregada (Decode)
  4 - Se a instrução utiliza mais parâmetros, busca por fetch sucessivos
  5 - Se a instrução realiza acesso à memória (leitura ou escrita), efetua o acesso
  6 - Executa a instrução
  7 - Vai para a etapa 1 para iniciar a execução da próxima instrução

Exemplos de Linguagens de Máquina

EXEMPLO DE ASSEMBLY DE UM PROCESSADOR HIPOTÉTICO

OBS: PROCESSADOR DE 8 BITs E ENDEREÇAMENTO DE 16 BITs

OPERAÇÕES COM INTEIROS

ADD	1-2 (1 ou 2 bytes)	
SUB	1-2	
AND	1-2	*OPERANDO*:
IOR	1-2	- Registrador (1 byte)
XOR	1-2	- Constante (2 bytes)
COMP	1	
CLR	1	*FORMATO*:
INC	1	Instrução *Destino*, *Origem*
DEC	1	

OPERAÇÕES DE DESLOCAMENTO

RL	1 (Rotate Left Acumulador)
RR	1 (Rotate Right Acumulador)

OPERAÇÕES DE FLUXO DE DADOS

MOV	1 (Acumulador ← → Registrador)
MOVC	1 (Acumulador ← Constante)
PUSH	1 (Insere na Pilha)
POP	1 (Retira da Pilha)
LOAD	3 (Memória: Endereço 2 bytes)
SAVE	3 (Memória: Endereço 2 bytes)
IN	3 (Dispositivos I/O: Endereço 2 bytes)
OUT	3 (Dispositivos I/O: Endereço 2 bytes)

BITs DE ESTADO (do Acumulador)

Z	(Zero): = 1 se Acumulador = 0
C	(Carry): = se Vai-um após soma

REGISTRADORES ESPECIAIS

ACC	(Acumulador)
PC	(Program Counter)
IR	(Instruction Register)
SP	(Stack Pointer)
MAR	(Memory Address Reg)
MDR	(Memory Data Reg)
IND	(Indirect Address Reg)

REGISTRADORES DE DADOS

R0 a R7 Registradores de uso geral

OPERAÇÕES DE FLUXO DE CONTROLE

GOTO	3 (Endereço 2 bytes)
CALL	3
RET	1 (Return)
	Condicionais
JZ	3 (Jump on Zero)
JNZ	3 (Jump on Non-Zero)
JC	3 (Jump on Carry)
JNC	3 (Jump in Non-Carry)
NOP	1 (No Operation)

INSTRUÇÕES DE MÁQUINA: Exemplos Reais

Loads

LDSB ADDR,DST	Load signed byte (8 bits)
LDUB ADDR,DST	Load unsigned byte (8 bits)
LDSH ADDR,DST	Load signed halfword (16 bits)
LDUH ADDR,DST	Load unsigned halfword (16)
LDSW ADDR,DST	Load signed word (32 bits)
LDUW ADDR,DST	Load unsigned word (32 bits)
LDX ADDR,DST	Load extended (64-bits)

Stores

STB SRC,ADDR	Store byte (8 bits)
STH SRC,ADDR	Store halfword (16 bits)
STW SRC,ADDR	Store word (32 bits)
STX SRC,ADDR	Store extended (64 bits)

Arithmetic

ADD R1,S2,DST	Add
ADDCC "	Add and set icc
ADDC "	Add with carry
ADDCCC "	Add with carry and set icc
SUB R1,S2,DST	Subtract
SUBCC "	Subtract and set icc
SUBC "	Subtract with carry
SUBCCC "	Subtract with carry and set icc
MULX R1,S2,DST	Multiply
SDIVX R1,S2,DST	Signed divide
UDIVX R1,S2,DST	Unsigned divide
TADCC R1,S2,DST	Tagged add

Shifts/rotates

SLL R1,S2,DST	Shift left logical (32 bits)
SLLX R1,S2,DST	Shift left logical extended (64)
SRL R1,S2,DST	Shift right logical (32 bits)
SRLX R1,S2,DST	Shift right logical extended (64)
SRA R1,S2,DST	Shift right arithmetic (32 bits)
SRAX R1,S2,DST	Shift right arithmetic ext. (64)

Boolean

AND R1,S2,DST	Boolean AND
ANDCC "	Boolean AND and set icc
ANDN "	Boolean NAND
ANDNCC "	Boolean NAND and set icc
OR R1,S2,DST	Boolean OR
ORCC "	Boolean OR and set icc
ORN "	Boolean NOR
ORNCC "	Boolean NOR and set icc
XOR R1,S2,DST	Boolean XOR
XORCC "	Boolean XOR and set icc
XNOR "	Boolean EXCLUSIVE NOR
XNORCC "	Boolean EXCL. NOR and set icc

Transfer of control

BPcc ADDR	Branch with prediction
BPr SRC,ADDR	Branch on register
CALL ADDR	Call procedure
RETURN ADDR	Return from procedure
JMPL ADDR,DST	Jump and Link
SAVE R1,S2,DST	Advance register windows
RESTORE "	Restore register windows
Tcc CC,TRAP#	Trap on condition
PREFETCH FCN	Prefetch data from memory
LDSTUB ADDR,R	Atomic load/store
MEMBAR MASK	Memory barrier

Miscellaneous

SETHI CON,DST	Set bits 10 to 31
MOVcc CC,S2,DST	Move on condition
MOVr R1,S2,DST	Move on register
NOP	No operation
POPC S1,DST	Population count
RDCCR V,DST	Read condition code register
WRCCR R1,S2,V	Write condition code register
RDPC V,DST	Read program counter

SRC = source register
DST = destination register
R1 = source register
S2 = source: register or immediate
ADDR = memory address

TRAP# = trap number
FCN = function code
MASK = operation type
CON = constant
V = register designator

CC = condition code set
R =destination register

cc = condition
r = LZ,LEZ,Z,NZ,GZ,GEZ

Figure 5-34. The primary UltraSPARC II integer instructions.

Loads

typeLOAD IND8	Push local variable onto stack
typeALOAD	Push array element on stack
BALOAD	Push byte from an array on stack
SALOAD	Push short from an array on stack
CALOAD	Push char from an array on stack
AALOAD	Push pointer from an array on "

Stores

typeSTORE IND8	Pop value and store in local var
typeASTORE	Pop value and store in array
BASTORE	Pop byte and store in array
SASTORE	Pop short and store in array
CASTORE	Pop char and store in array
AASTORE	Pop pointer and store in array

Pushes

BIPUSH CON8	Push a small constant on stack
SIPUSH CON16	Push 16-bit constant on stack
LDC IND8	Push constant from const pool
typeCONST_#	Push immediate constant
ACONST_NULL	Push a null pointer on stack

Arithmetic

typeADD	Add
typeSUB	Subtract
typeMUL	Multiple
typeDIV	Divide
typeREM	Remainder
typeNEG	Negate

Boolean/shift

iIAND	Boolean AND
iIOR	Boolean OR
iIXOR	Boolean EXCLUSIVE OR
iISHL	Shift left
iISHR	Shift right
iIUSHR	Unsigned shift right

Conversion

x2y	Convert x to y
i2c	Convert integer to char
i2b	Convert integer to byte

Stack management

DUPxx	Six instructions for duping
POP	Pop an int from stk and discard
POP2	Pop two ints from stk and discard
SWAP	Swap top two ints on stack

Comparison

IF_ICMPrel OFFSET16	Conditional branch
IF_ACMPEQ OFFSET16	Branch if two ptrs equal
IF_ACMPNE OFFSET16	Branch if ptrs unequal
IFrel OFFSET16	Test 1 value and branch
IFNULL OFFSET16	Branch if ptr is null
IFNONNULL OFFSET16	Branch if ptr is nonnull
LCMP	Compare two longs
FCMPL	Compare 2 floats for <
FCMPG	Compare 2 floats for >
DCMPL	Compare doubles for <
DCMPG	Compare doubles for >

Transfer of control

INVOKEVIRTUAL IND16	Method invocation
INVOKESTATIC IND16	Method invocation
INVOKEINTERFACE ...	Method invocation
INVOKESPECIAL IND16	Method invocation
JSR OFFSET16	Invoke finally clause
typeRETURN	Return value
ARETURN	Return pointer
RETURN	Return void
RET IND8	Return from finally
GOTO OFFSET16	Unconditional branch

Arrays

ANEWARRAY IND16	Create array of ptrs
NEWARRAY ATYPE	Create array of atype
MULTINEWARRAY IN16,D	Create multidim array
ARRAYLENGTH	Get array length

Miscellaneous

IINC IND8,CON8	Increment local variable
WIDE	Wide prefix
NOP	No operation
GETFIELD IND16	Read field from object
PUTFIELD IND16	Write field to object
GETSTATIC IND16	Get static field from class
NEW IND16	Create a new object
INSTANCEOF OFFSET16	Determine type of obj
CHECKCAST IND16	Check object type
ATHROW	Throw exception
LOOKUPSWITCH ...	Sparse multiway branch
TABLESWITCH ...	Dense multiway branch
MONITORENTER	Enter a monitor
MONITOREXIT	Leave a monitor

IND8/16 = index of local variable
CON8/16, D, ATYPE = constant

type, x, y = I, L, F, D
OFFSET16 for branch

The JVM instruction set.

PROGRAMAÇÃO ASSEMBLER

- Simulator genérico
  https://schweigi.github.io/assembler-simulator/
- Editores Assembrer 6502:
  http://www.asm80.com/
  http://skilldrick.github.io/easy6502/
- Compilador 8080 e z80:
  http://asdasd.rpg.fi/~svo/i8080/
  http://clrhome.org/asm/
- Compilador online x86
  https://www.tutorialspoint.com/compile_assembly_online.php

Exercício

EXERCÍCIO DE PROGRAMAÇÃO EM ASSEMBLY

Utilizando o assembly 6502 (http://www.6502.org/tutorials/6502opcodes.html) e o emulador da página anterior, desenvolver em linguagem de máquina (Assembly) um programa que implementa um relógio digital.

Dados:
- um gerador de sinais no endereço F000, que gera pulsos a cada 1 segundo;
- um display no endereço F001, que apresenta as horas;
- um display no endereço F002, que apresenta os minutos;
- um display no endereço F003, que apresenta os segundos.

Utilizar rótulos (labels) para identificar posições de endereços no programa.
O gerador de sinais e displays são dispositivos de E/S (I/O)