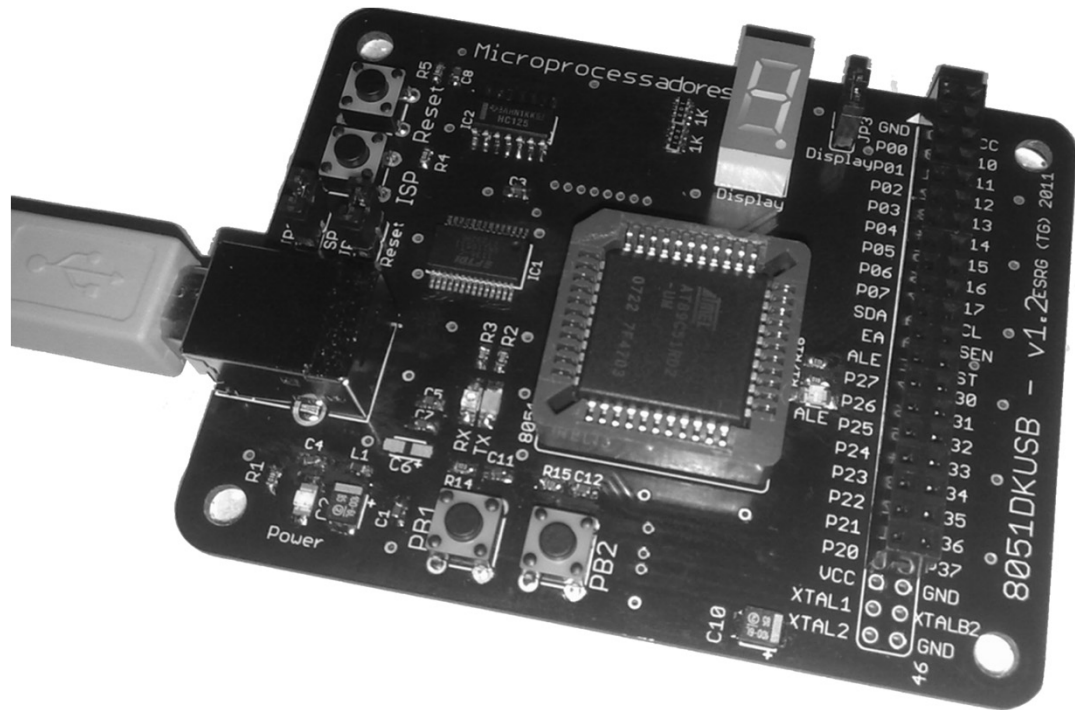
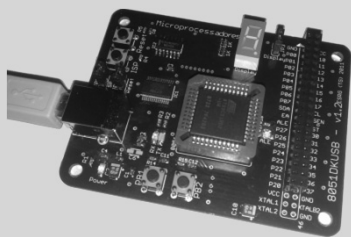


Mestrado Integrado em Eng. Electrónica Industrial e Computadores



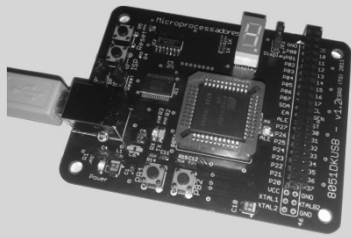
**Sistema
Computadorizado**

Microcontroladores
2º Ano – A03

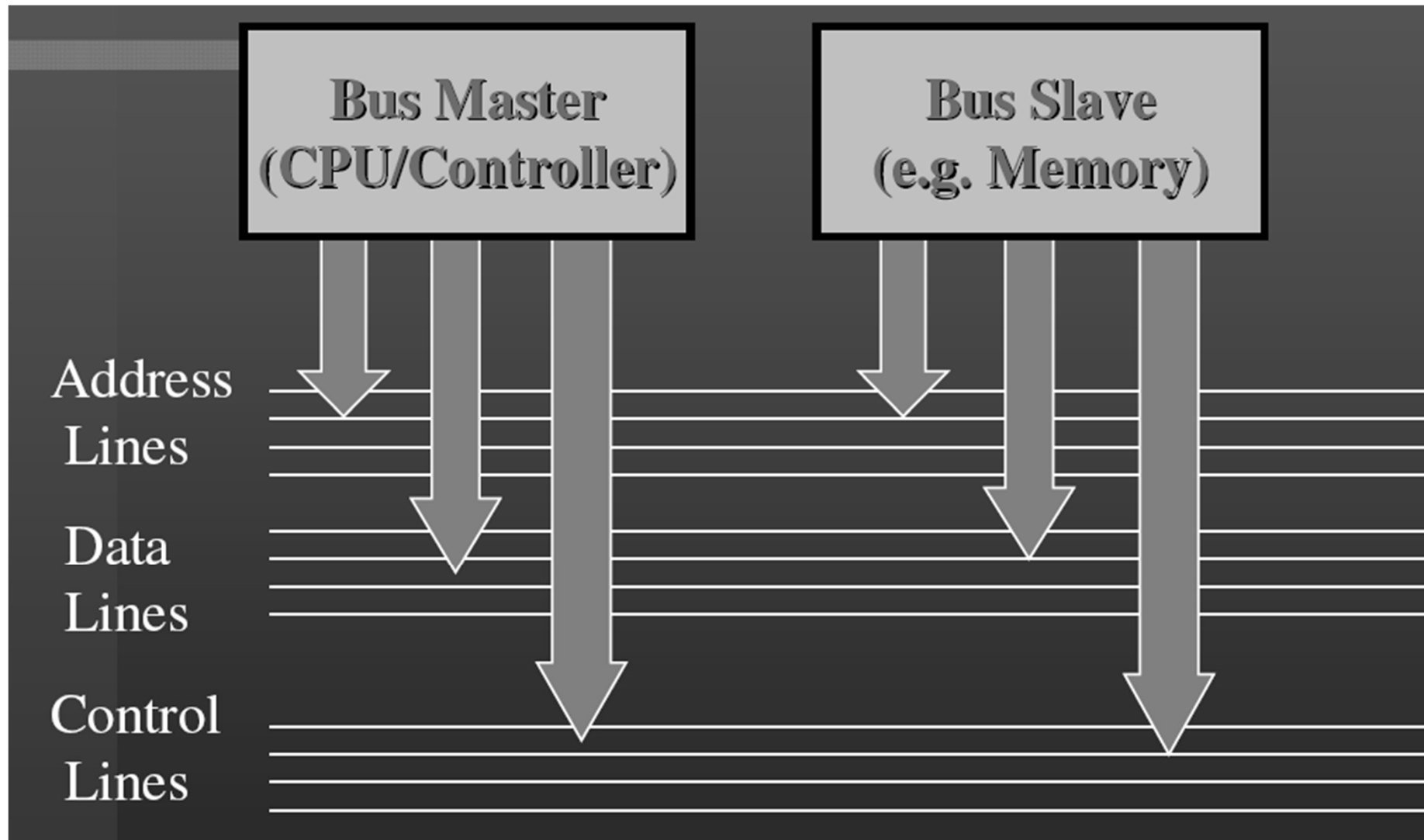


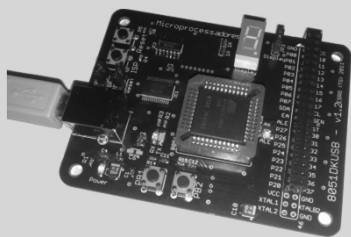
Barramentos

- O processador de um computador comunica com outros módulos do computador através de um dispositivo chamado barramento (bus).
- Existem várias arquiteturas de barramento disponíveis no mercado, tais como PCI; cPCI; VME ...
- Todas as arquiteturas incluem um barramento de controlo, um barramento de dados e um barramento de endereços.
- Similarmente os microcontroladores utilizam também uma arquitetura de barramento para comunicarem com todos os módulos existentes dentro do chip. A arquitetura pode variar de micro para micro, mas inclui os barramentos referidos.



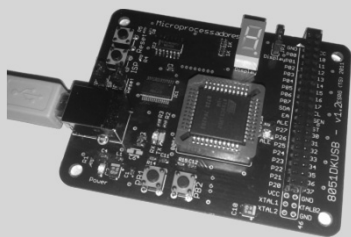
Barramentos





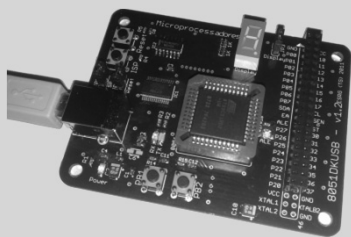
Sequência de Escrita (Write)

- O **master** coloca o endereço da memória, onde os dados devem ser escritos, no barramento de endereços e qualifica-os usando as linhas de controlo;
- O **master** sinaliza, usando as linhas de controlo, que esta é uma operação de escrita;
- O **master** coloca os dados a serem escritos no barramento de dados e qualifica-os utilizando as linhas de controlo;
- Após a qualificação do endereço pelo **master**, o **slave** compara o endereço no barramento de endereços com o seu próprio endereço. Se a operação de **Write** lhe é destinada, ele adquire os dados e sinaliza usando as linhas de controlo que terminou a operação (faz o *acknowledge* dos dados).



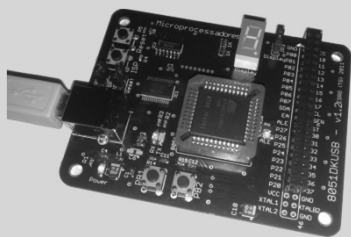
Sequência de leitura (Read)

- O **master** coloca o endereço de memória de onde pretende ler os dados no barramento e qualifica-o usando as linhas de controlo;
- O **master** sinaliza, usando as linhas de controlo, que se trata de uma sequência de leitura;
- O **master** sinaliza que está pronto a receber os dados usando as linhas de controlo;
- O **slave** compara o endereço no barramento com o seu próprio endereço, após o **master** o ter qualificado. Se a leitura se refere a ele, ele coloca os dados no barramento e sinaliza, usando as linhas de controlo que terminou (*acknowledge* dos dados). No final o **master** realiza o *Latch* dos dados.



Transferência Paralelo

- Num barramento paralelo todos os bits de um byte ou word (palavra: 16-bit ou 32-bit) são transferidos simultaneamente;
- Se tivermos um barramento com um byte de tamanho (8 linhas) e uma frequência de 1MHz então temos uma velocidade de barramento de 1Mbytes/seg;
- Num barramento há transferência de dados em paralelo (oposta à transferência de dados em série que iremos analisar posteriormente).



Revisões

- Qual é o formato de um programa na memória ?

- Consideremos o seguinte programa em C:

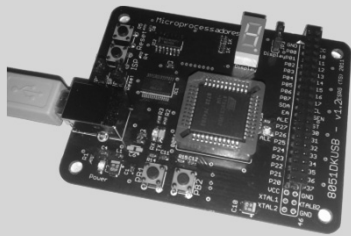
```
int max(int a, int b)
{
    if (a > b) return a;
    else return b;
}
```

- Mnemónicas das Instruções

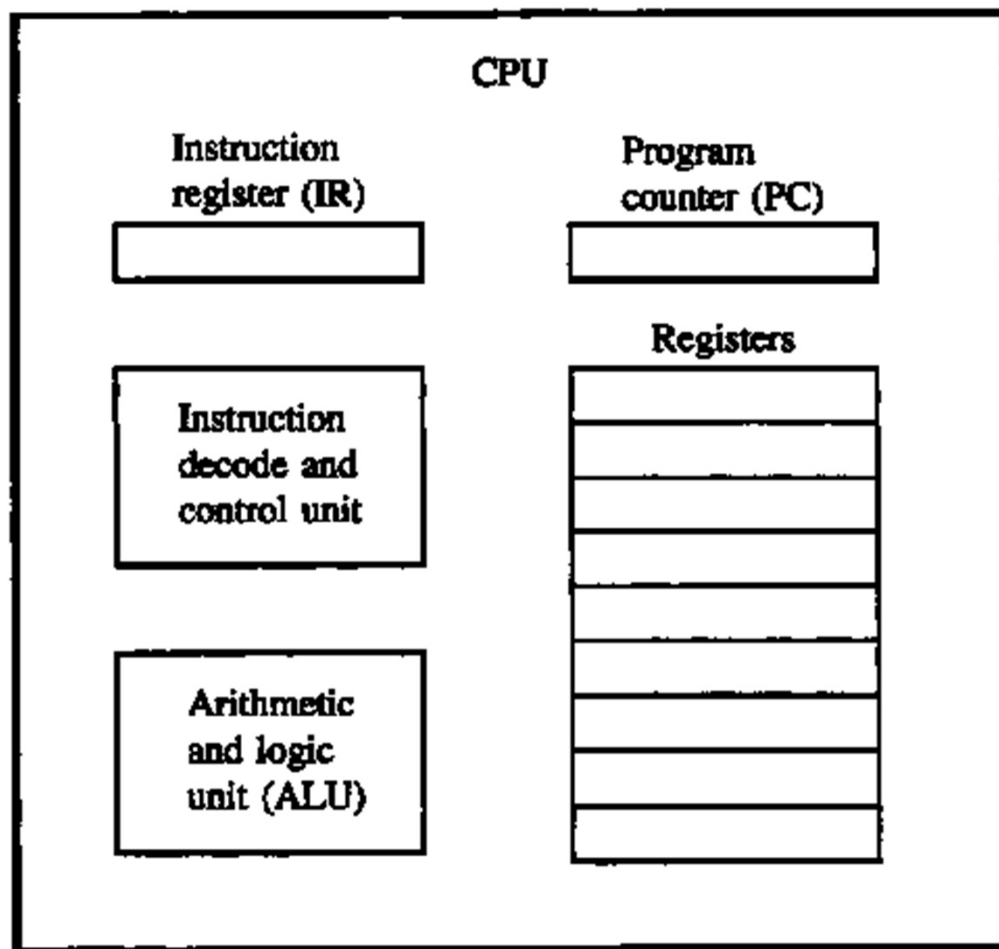
```
MOV R0,A
MOV R1,B
CLR C
SUBB A,R1
JC LBL1
MOV A,R0
MOV R3,A
RET
LBL1: MOV A,R1
      MOV R3,A
      RET
```

- Código máquina
90A7CB89210376

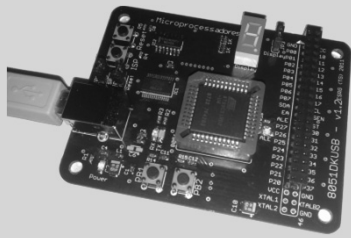
Programa armazenado na memória (RAM / ROM)



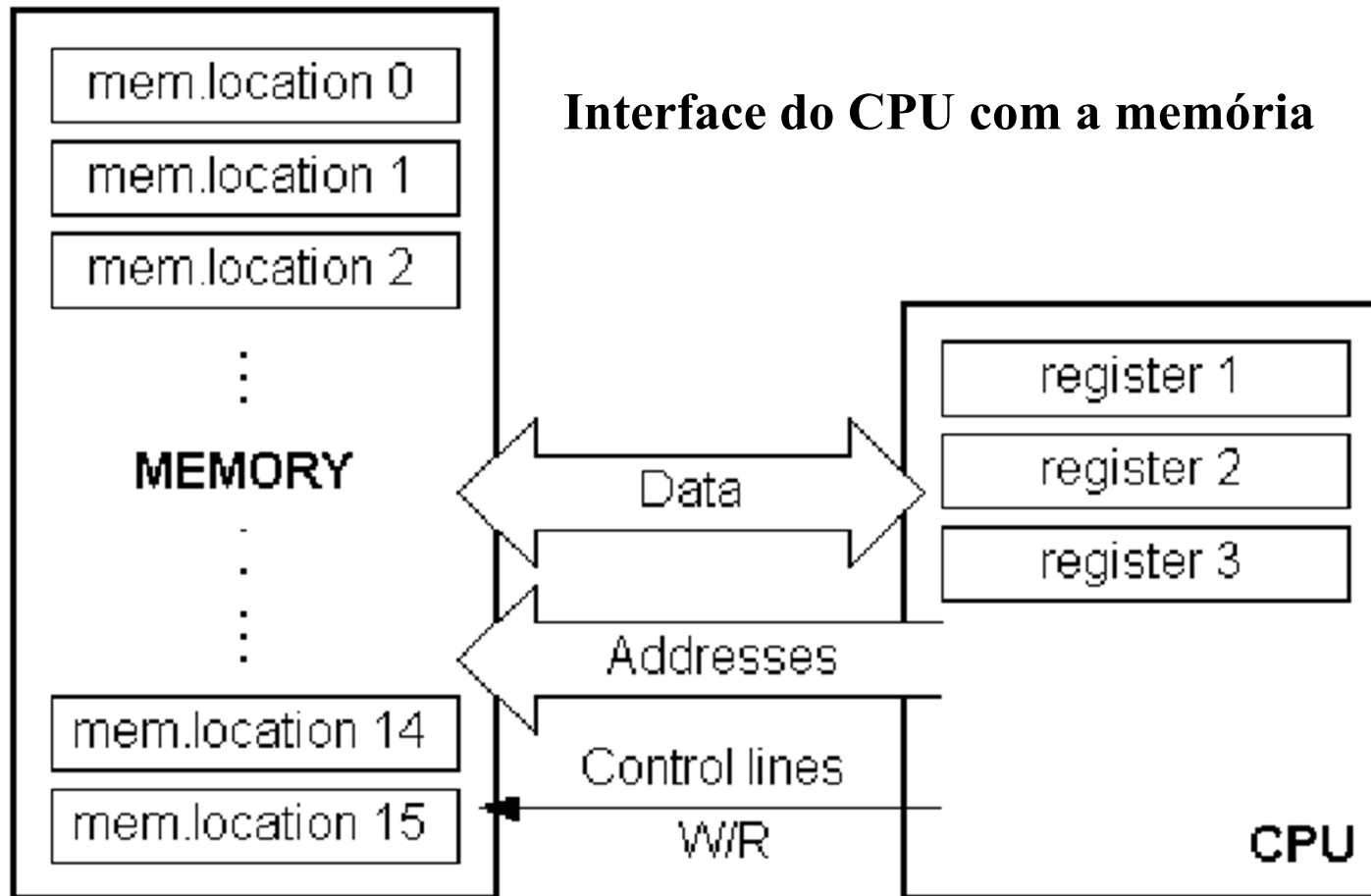
CPU

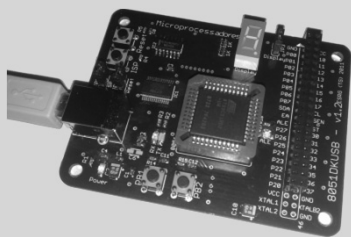


Exemplo de uma unidade de processamento central

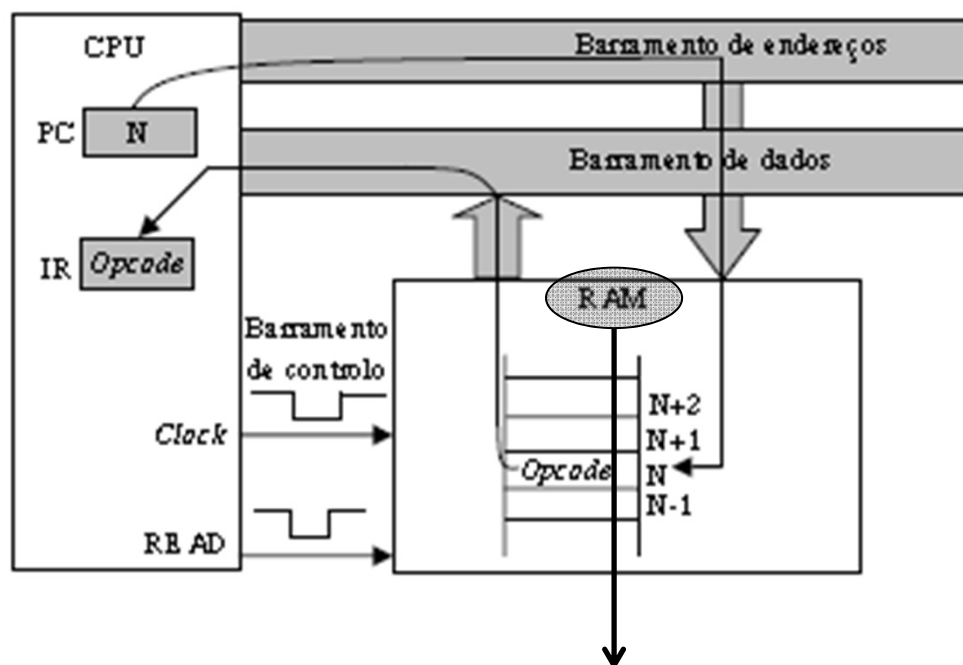


Bus - barramento





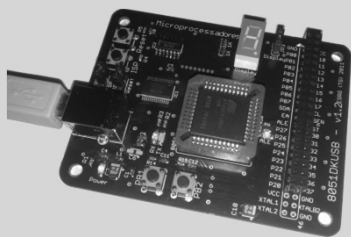
Fetch de um opcode



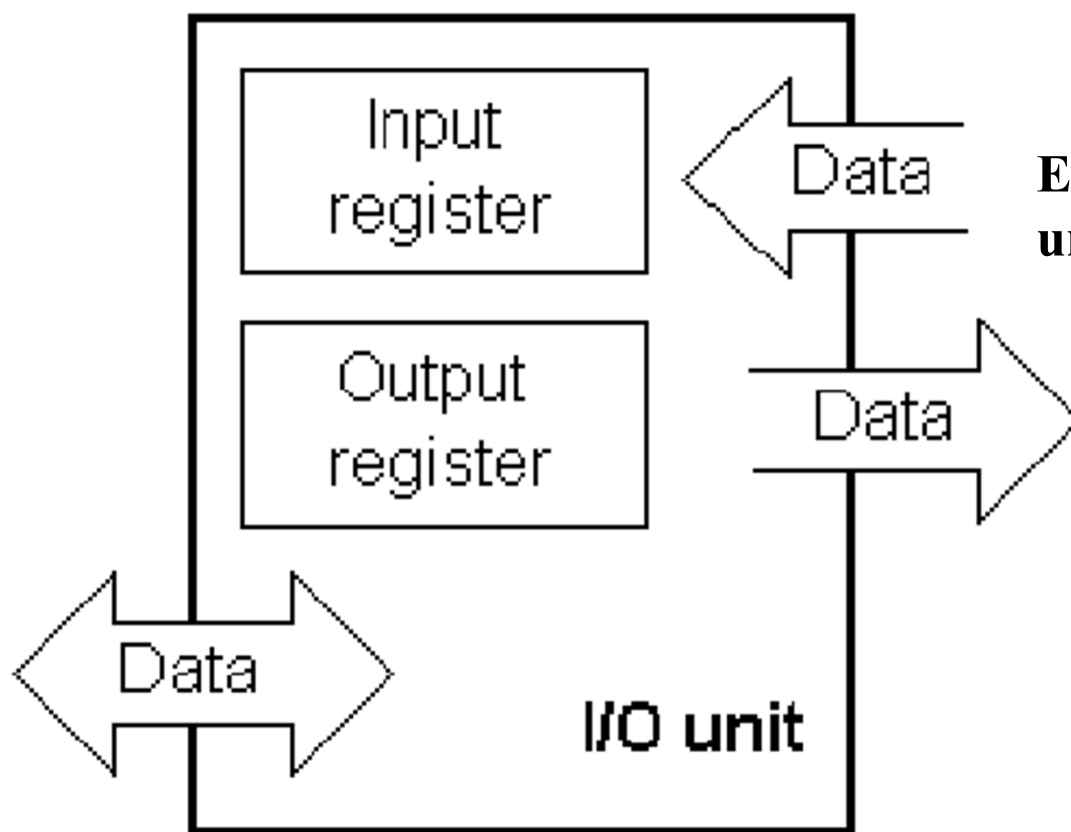
Passos

1. O conteúdo do PC é colocado no barramento de endereços;
2. O PC é incrementado para o *fetch* da próxima instrução.
3. A linha de controlo READ é activada;
4. A memória lê o conteúdo do endereço fornecido (código da instrução/*opcode*) e coloca-o no barramento de dados;
5. O opcode da instrução é transferido do barramento de dados para o registo interno IR;

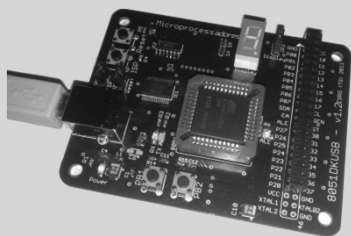
No caso de um microcontrolador esta memória é de que tipo?



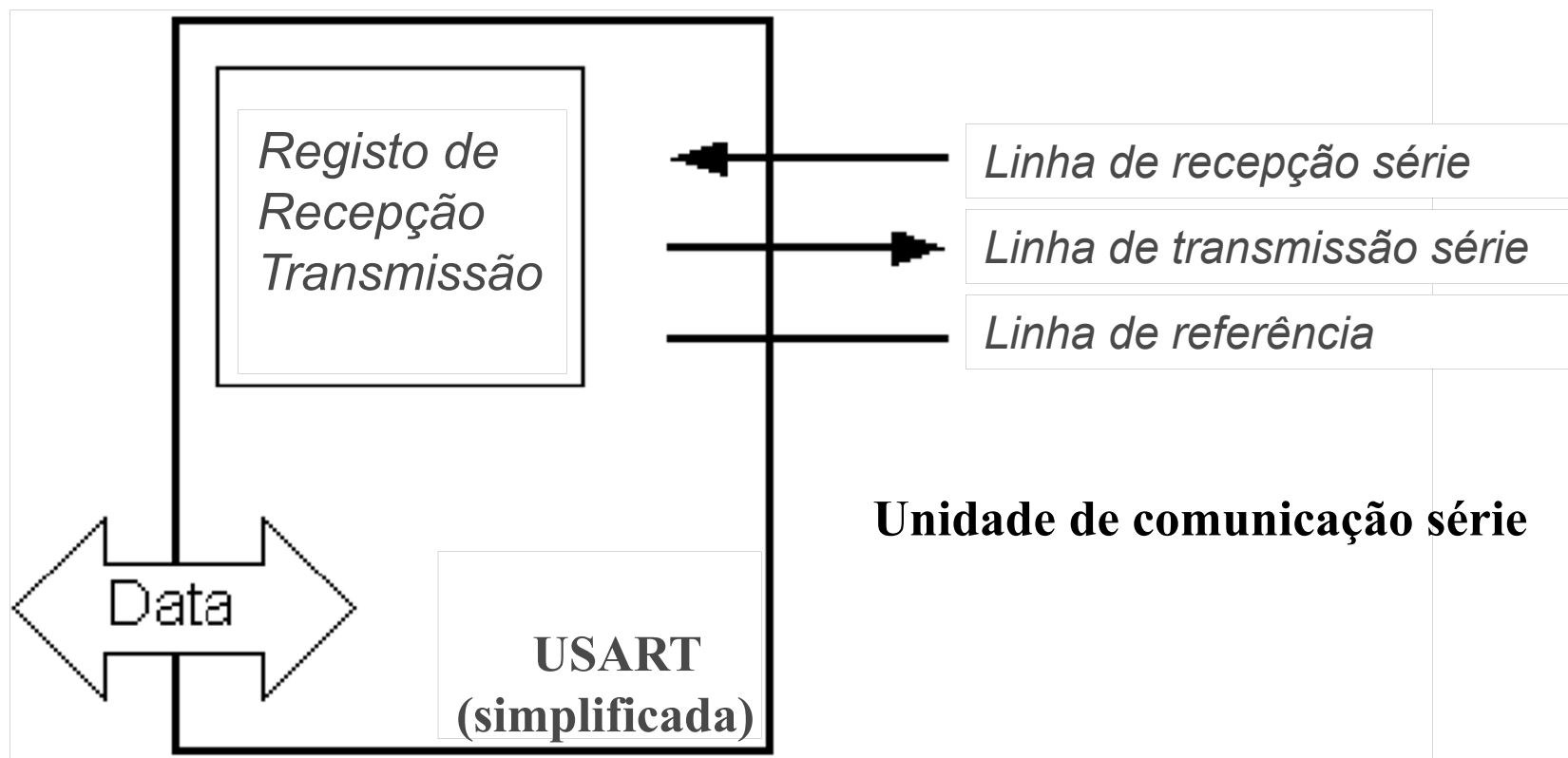
Unidade de Input/Output

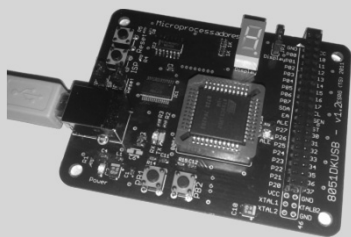


**Exemplo simplificado da
unidade de I/O (entrada/saída)**

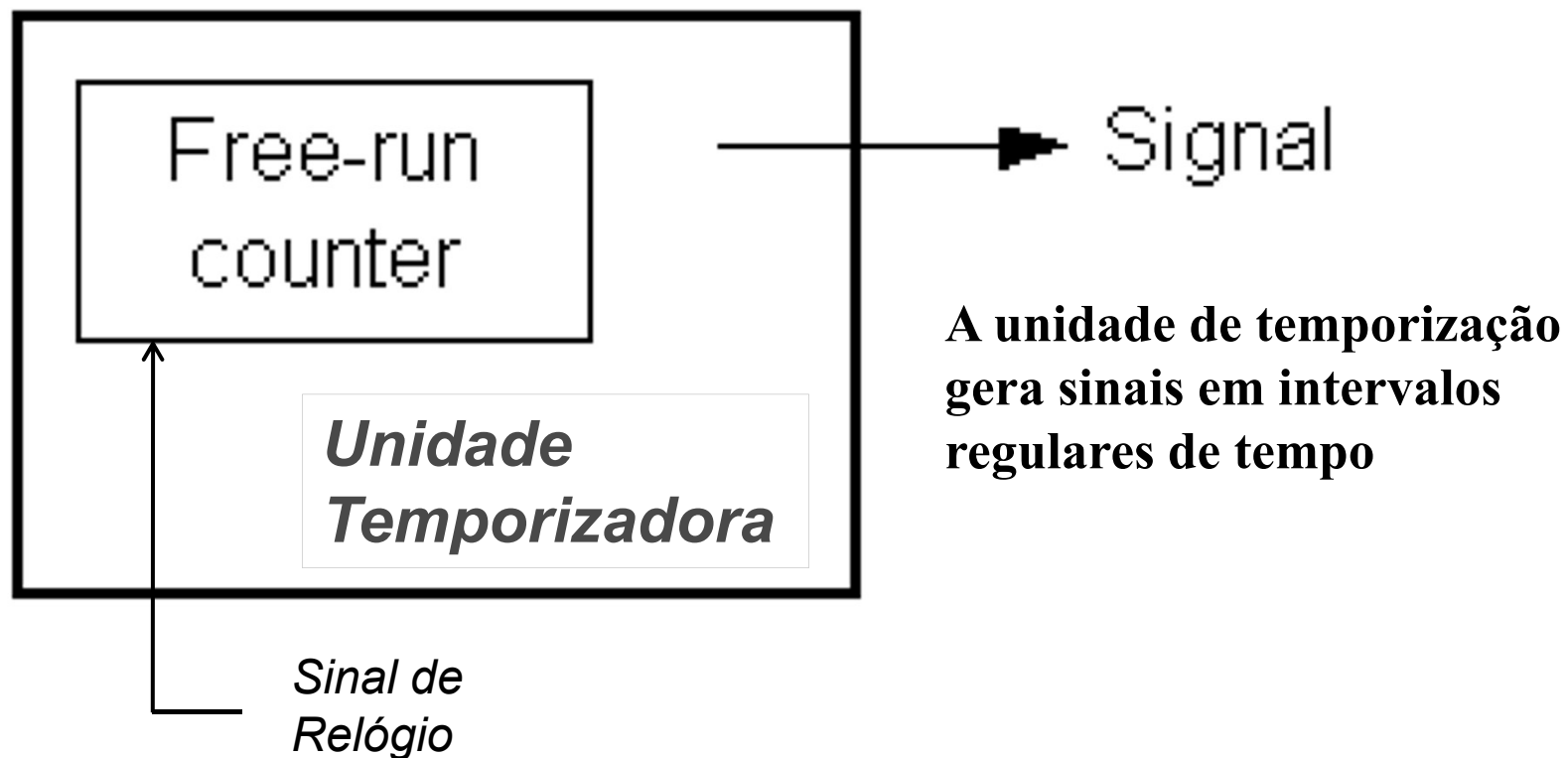


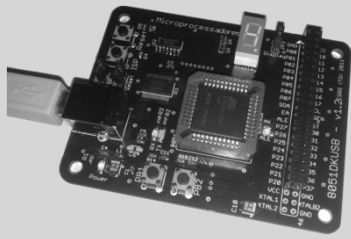
Unidade de Comunicação





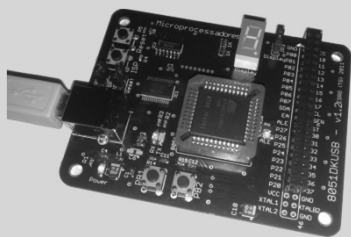
Unidade de temporização





Unidade de Watchdog

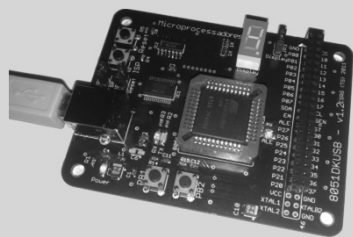




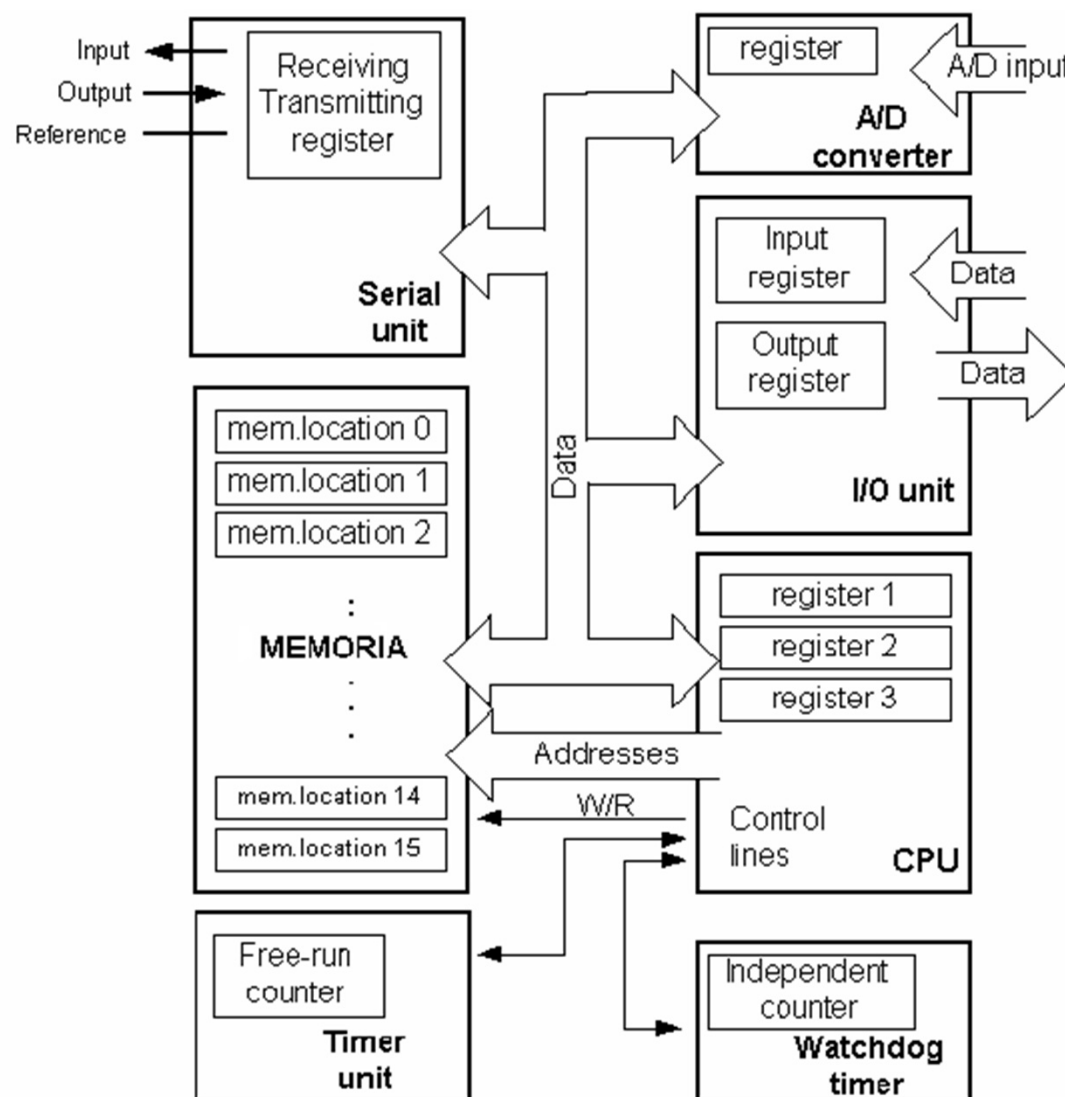
Unidade de Conversão A/D

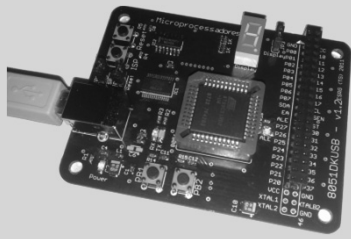


Bloco de conversão de um sinal analógico numa representação digital

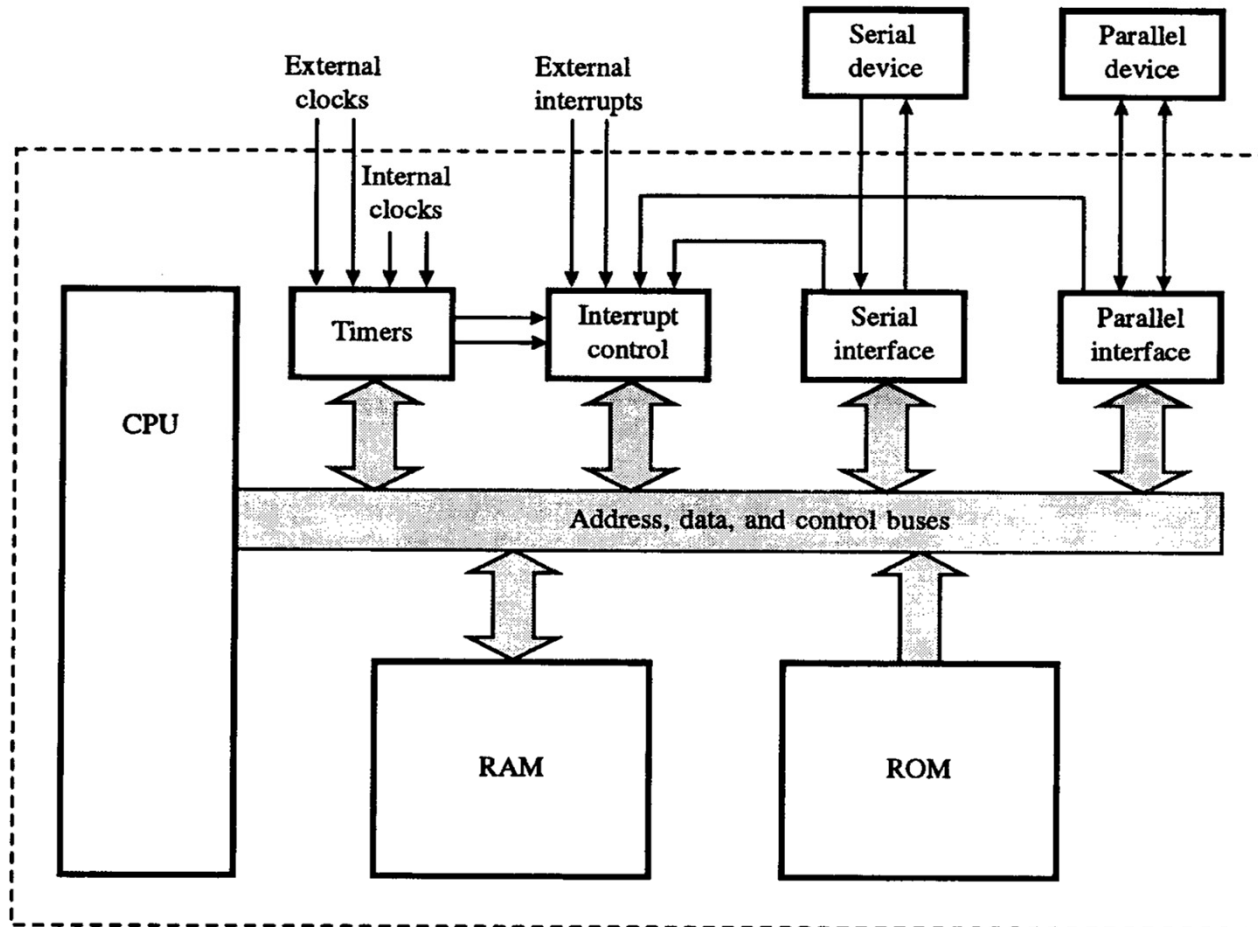


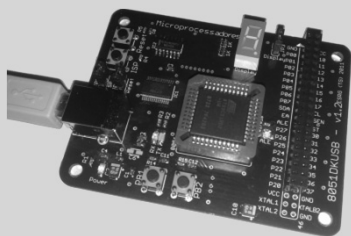
Estrutura de um microcontrolador





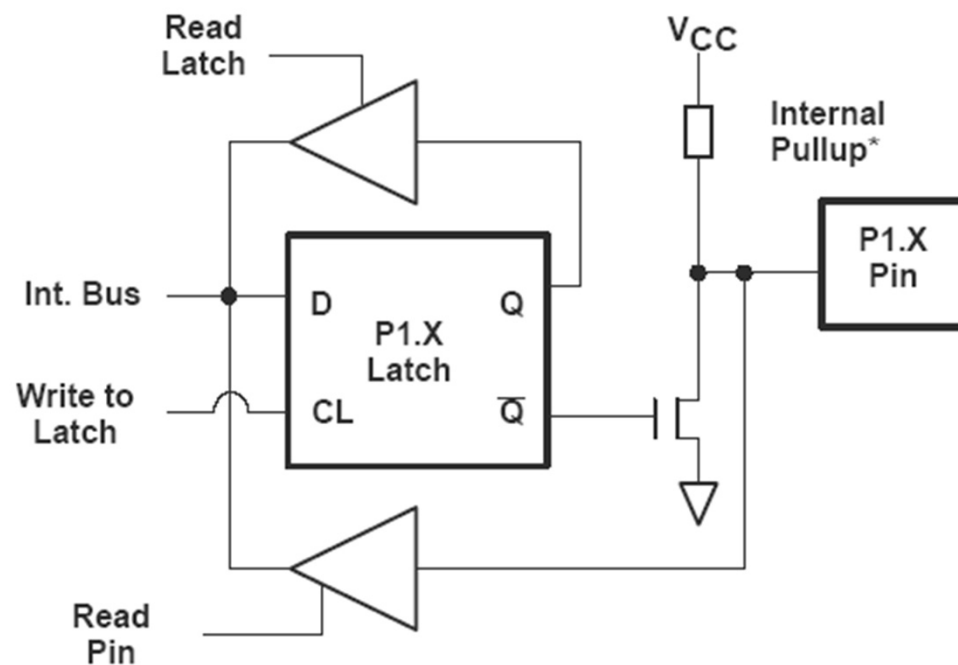
Estrutura genérica

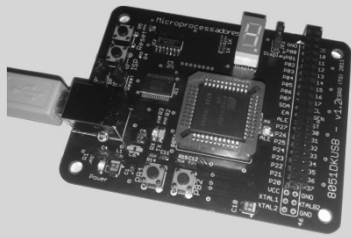




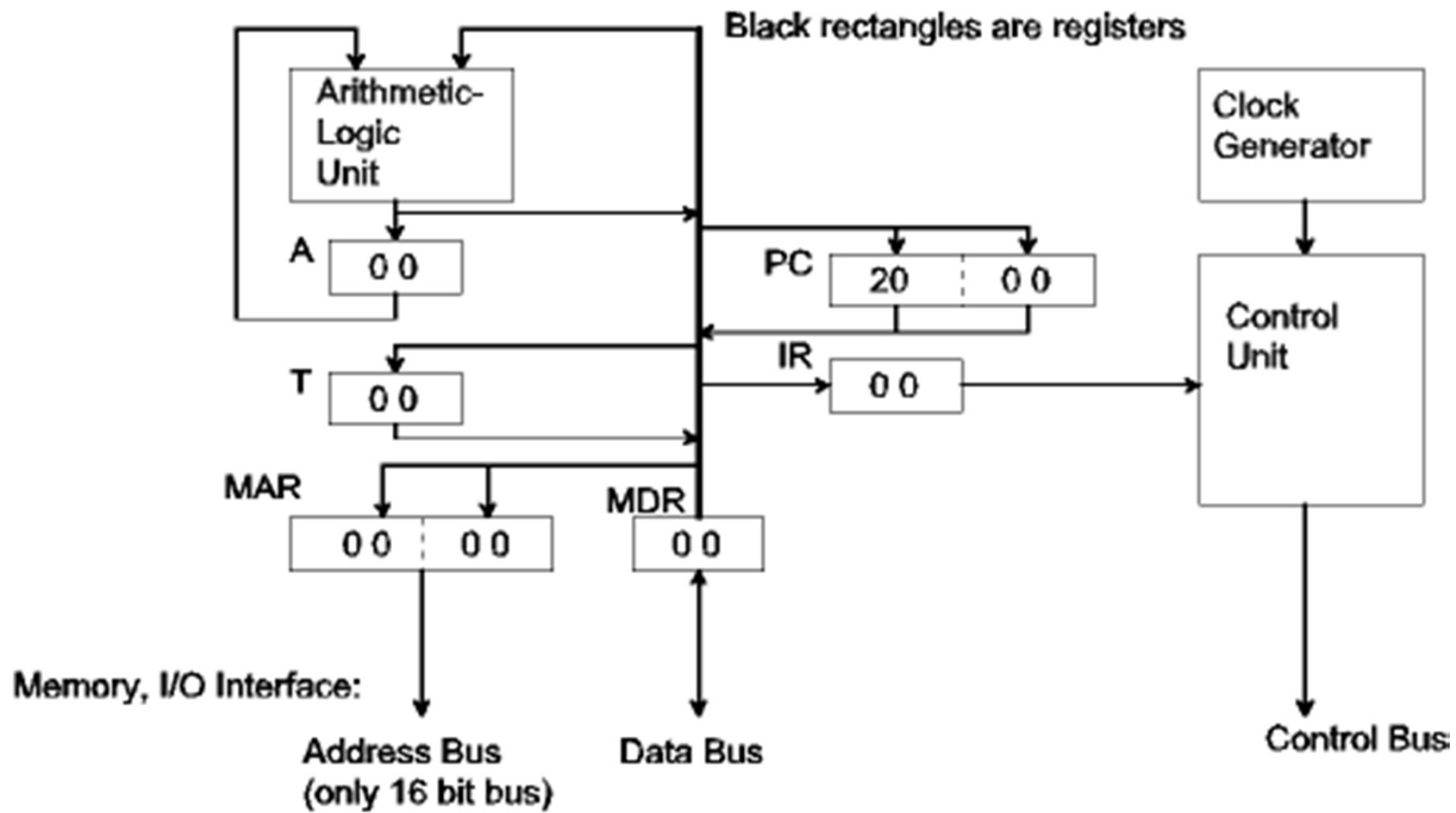
Portos de Entrada/Saída (I/O)

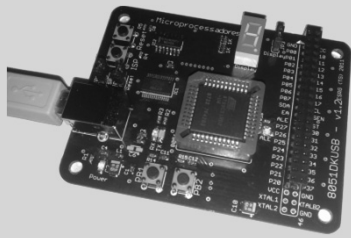
- A saída de um flip-flop (Latch) pode ser ligada directamente aos pinos de um microcontrolador, tornando possível a leitura do valor digital presente no pino ou a escrita de um valor no pino.
- Há registos (8 Latches) que estão directamente conectados aos pinos de um micro. São designados Portos de Entrada/Saída (Portos de I/O).
- Para que um CPU controle o porto de I/O este precisa de estar conectado aos barramentos. O modo mais simples de o conectar é fazer com que as Latches do porto estejam mapeadas em memória, permitindo assim operações de leitura/escrita no porto.
- De modo a simplificar o controlo dos portos (diminuindo o tamanho do bus de controlo) e como os pinos individuais dos portos apenas podem ser, num dado instante, de entrada ou de saída (binário), quase todos os microcontroladores utilizam um registo adicional (também mapeado em memória) que permite definir se um pino do porto é entrada ou saída de dados.



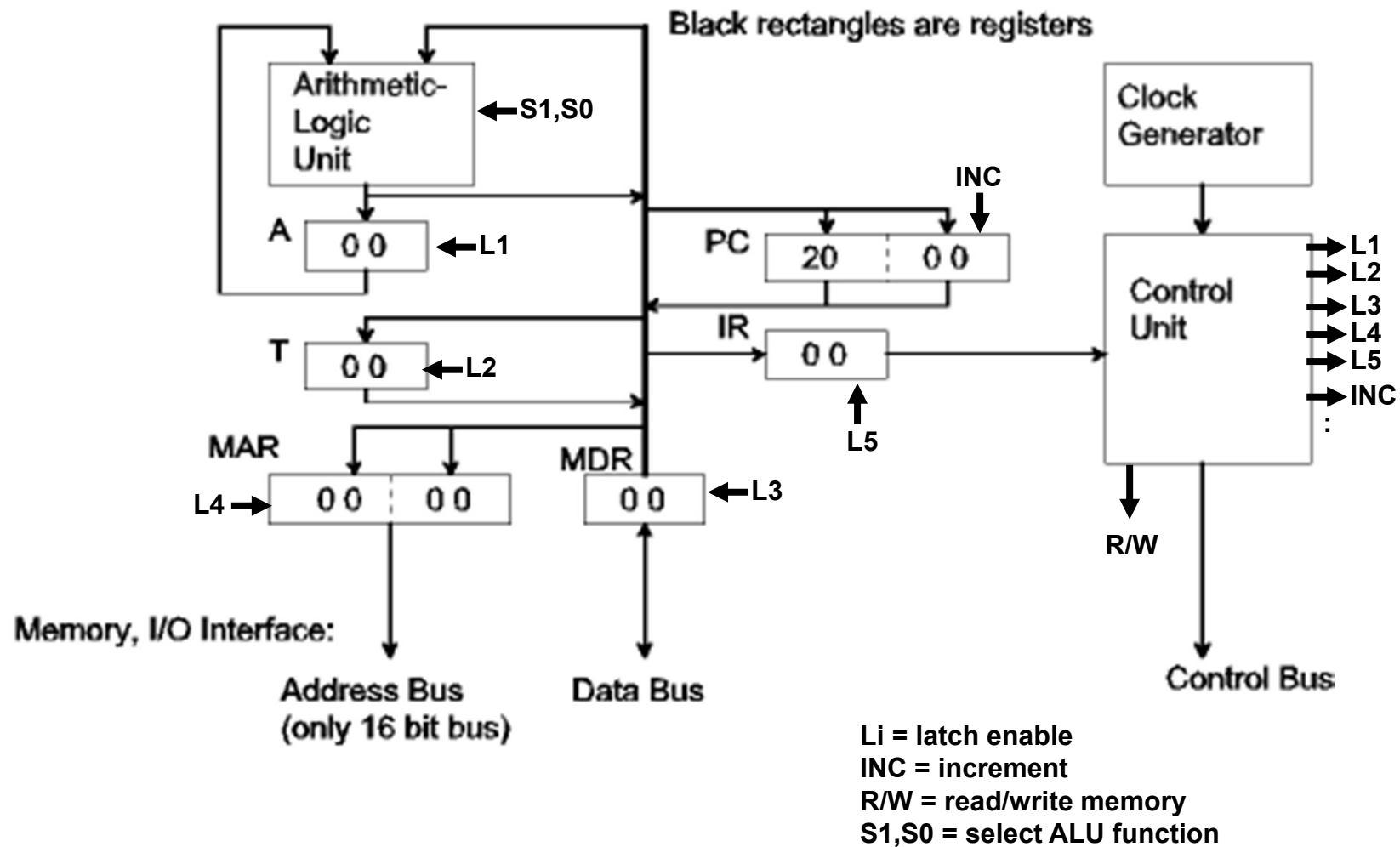


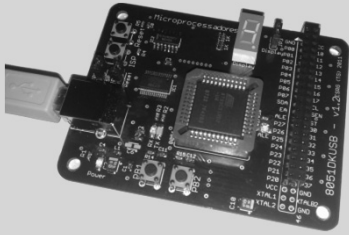
Exemplo de um processador





Sinais de controle

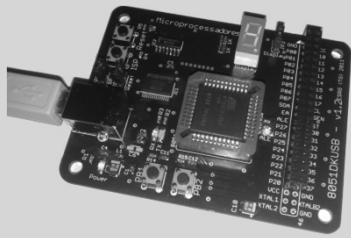




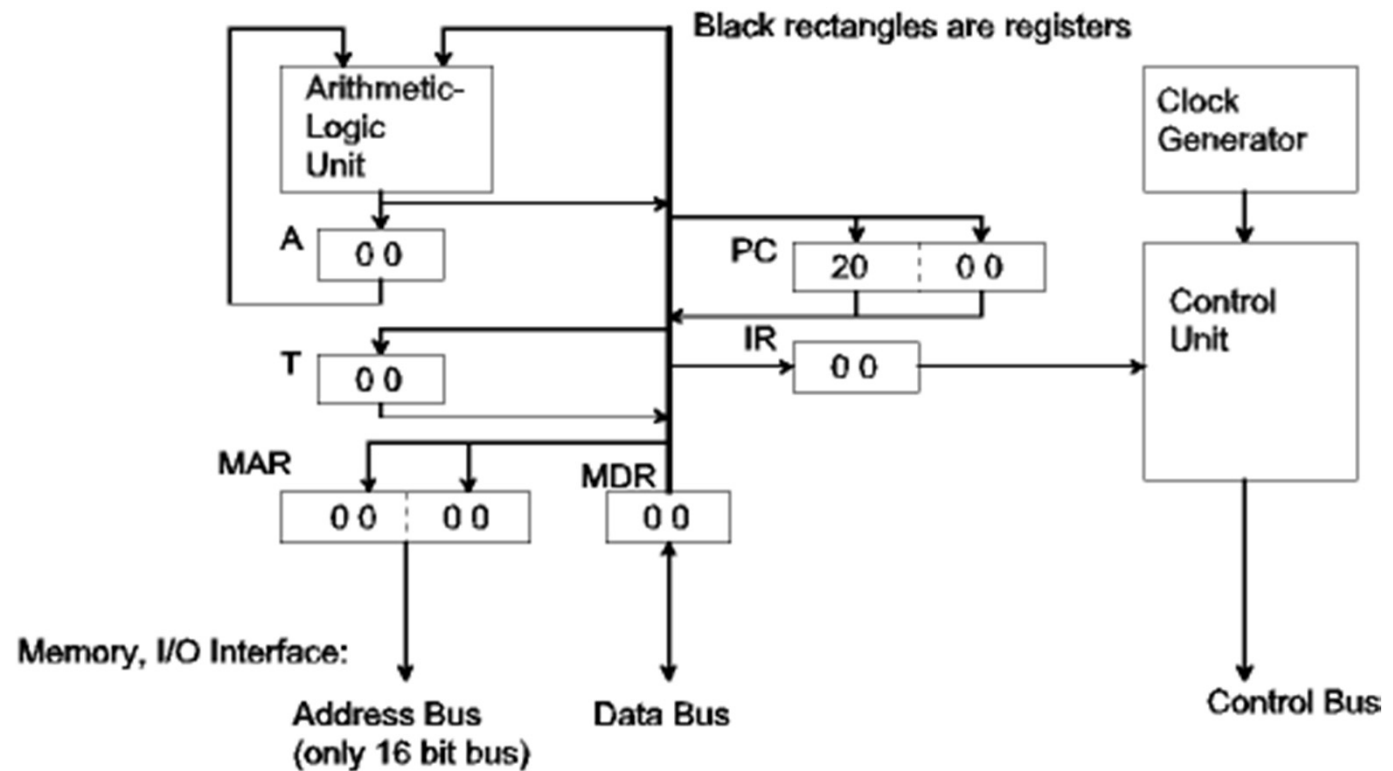
Conteúdo de uma memória

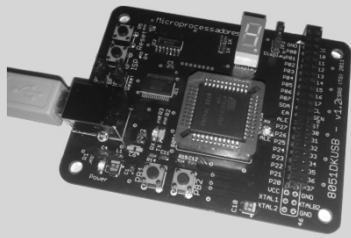
Location	Contents	Instruction
\$2000	\$B6	LDA \$1000
\$2001	\$10	
\$2002	\$00	
\$2003	\$BB	ADD \$1001
\$2004	\$10	
\$2005	\$01	
\$2006	\$43	DEC
\$2007	\$7A	STA \$1002
\$2008	\$10	
\$2009	\$02	

Address	Contents
\$1000	\$25
\$1001	\$37
\$1002	(unknown)



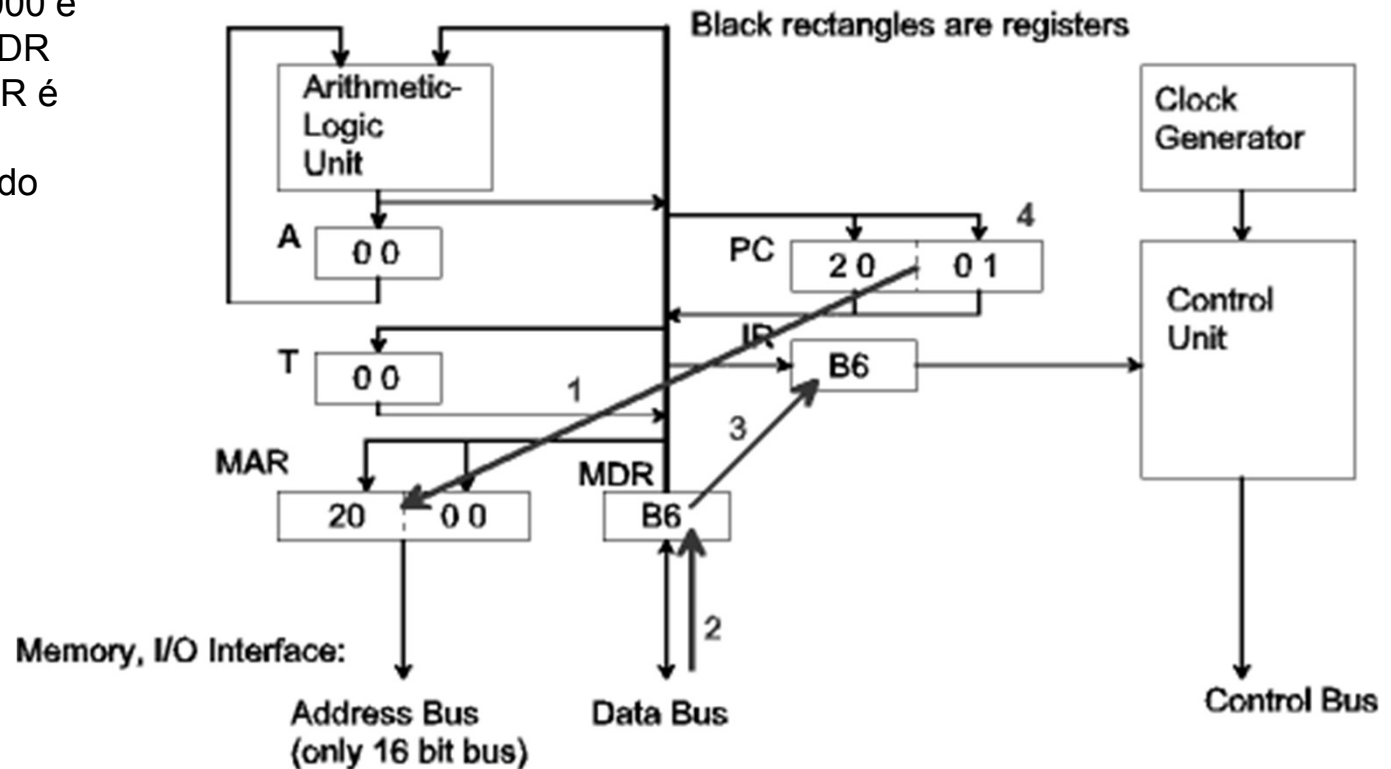
Execução de 'LDAA'

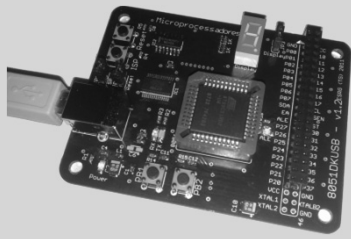




Execução de 'LDAA'

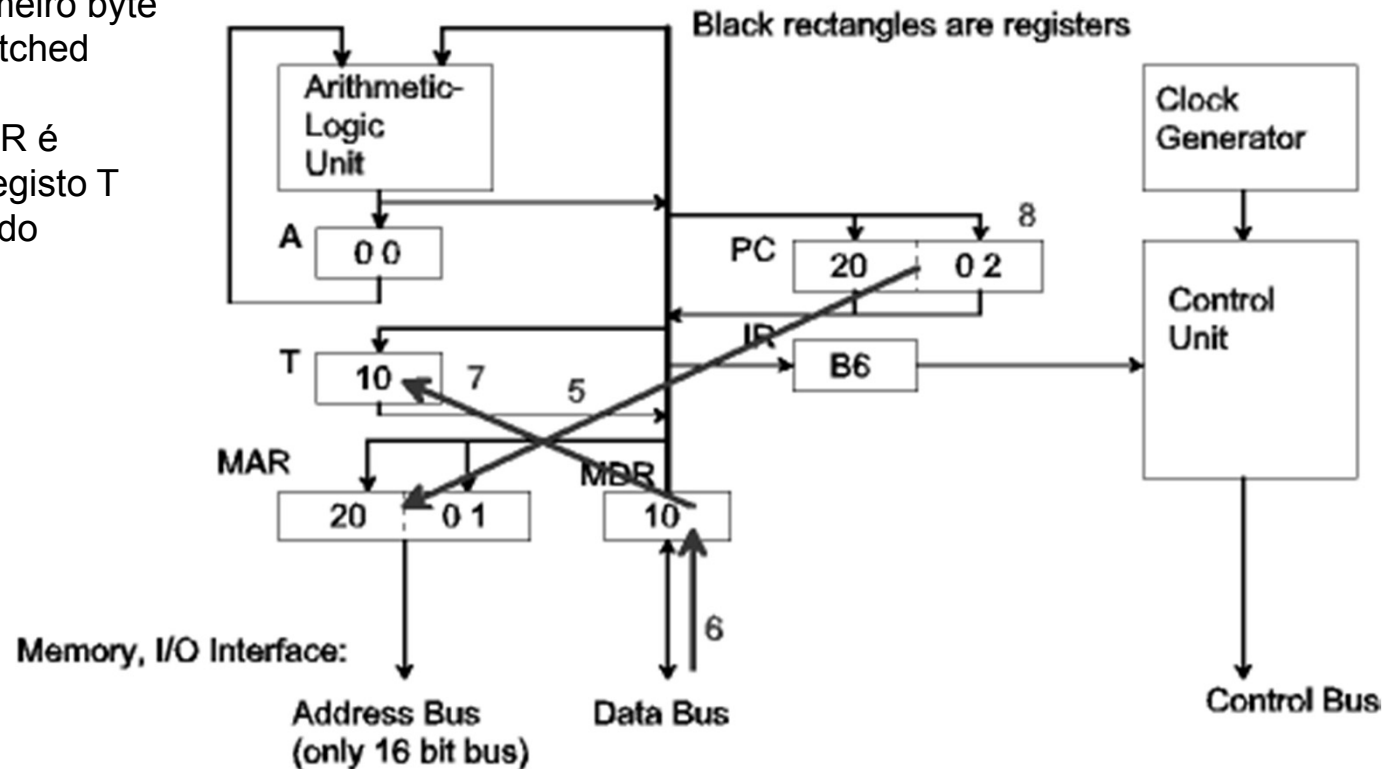
- (1) PC é copiado para o MAR
- (2) Conteúdo de \$2000 é latched para o MDR
- (3) Conteúdo de MDR é copiado para IR
- (4) PC é incrementado

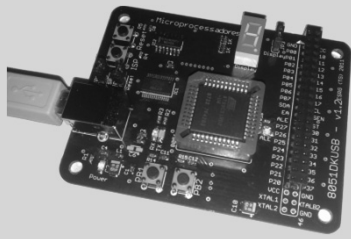




Execução de 'LDAA'

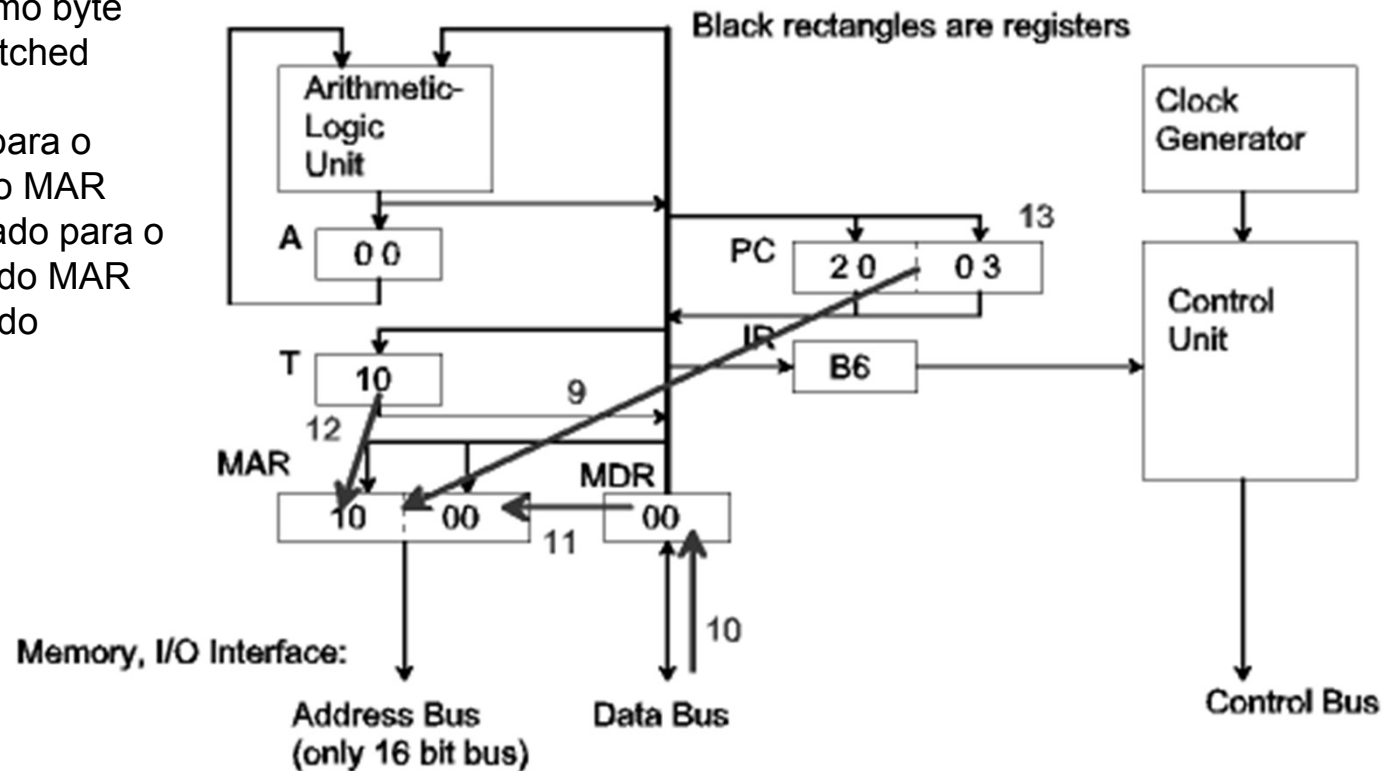
- (5) PC é copiado para o MAR
- (6) Conteúdo do primeiro byte do operando é latched para o MDR
- (7) Conteúdo do MDR é copiado para o registro T
- (8) PC é incrementado

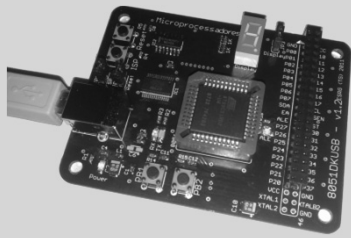




Execução de 'LDAA'

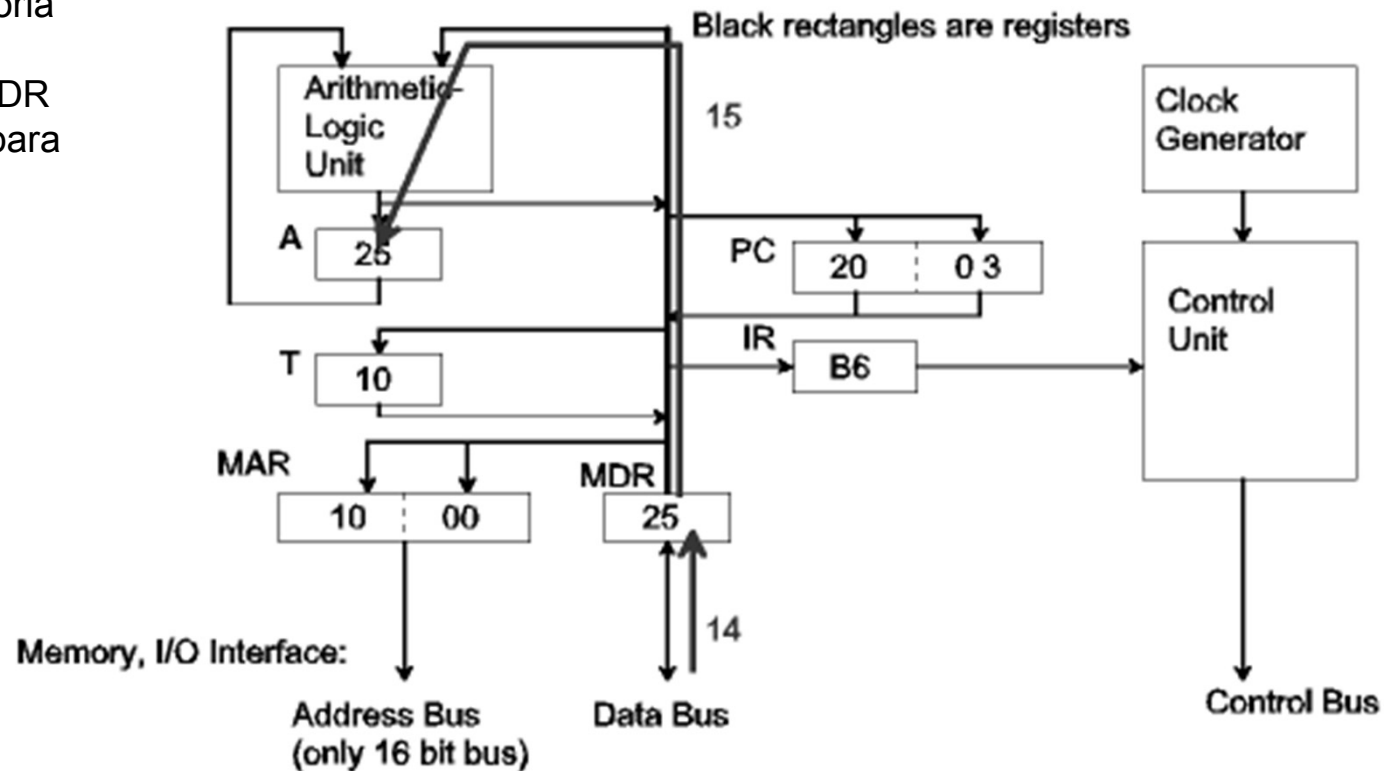
- (9) PC é copiado para o MAR
- (10) Conteúdo do último byte do operando é latched para o MDR
- (11) MDR é copiado para o LSB (low byte) do MAR
- (12) Registo T é copiado para o MSB (high byte) do MAR
- (13) PC é incrementado

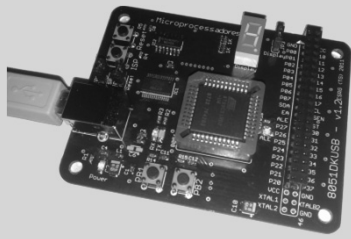




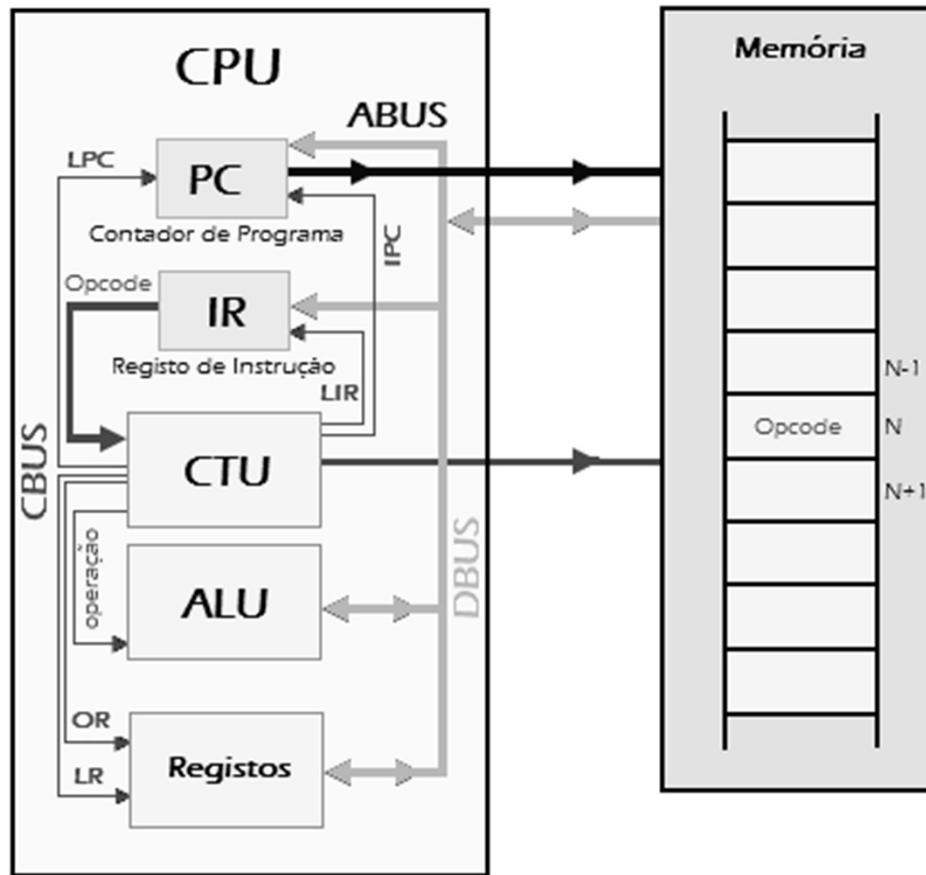
Execução de 'LDAA'

- (14) Byte de dados da posição de memória MAR (\$1000) é latched para o MDR
- (15) MDR é copiado para o acumulador A



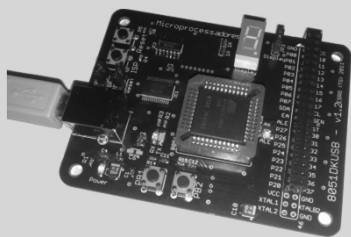


Fetch de um opcode



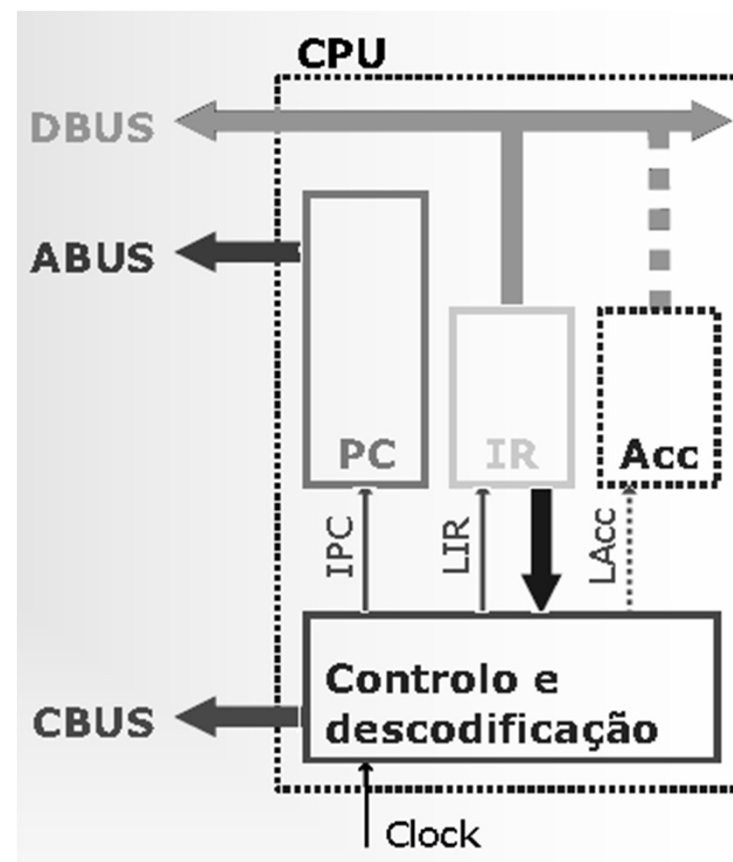
Passos

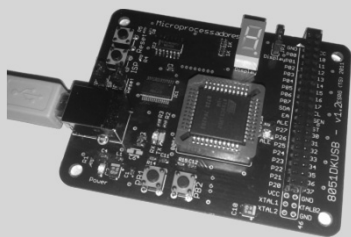
1. O conteúdo do PC é colocado no barramento de endereços;
2. O PC é incrementado para o *fetch* da próxima instrução;
3. A linha de controlo RD é activada;
4. A memória lê o conteúdo do endereço fornecido (código da instrução) e coloca-o no barramento de dados;
5. O *opcode* da instrução é transferido do barramento de dados para o registo interno IR;
6. O CTU descodifica o *opcode* e controla a execução das microinstruções a executar para realizar a operação.



Ler um *opcode*

- Leitura de uma instrução
 - O CPU envia à memória:
 - Endereço: ABUS;
 - Sinal de leitura: CBUS.
 - Recebe:
 - O *opcode* armazenado na posição ABUS em DBUS;
 - O *opcode* é armazenado no IR.
 - Hardware:
 - Registo de endereço – PC;
 - Registo de Instrução – IR;
 - Unidade de Controlo e de descodificação das instruções – CTU.
 - Sinais de Controlo:
 - IPC – Incrementar PC;
 - LIR – Load IR;
 - LAcc – Load Acumulador.





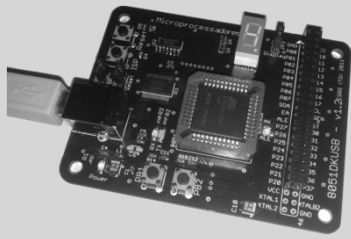
Executar uma instrução

- Colocar no Acumulador (A ou Acc) um valor constante:

Instrução no 8051 Opcode da Instrução

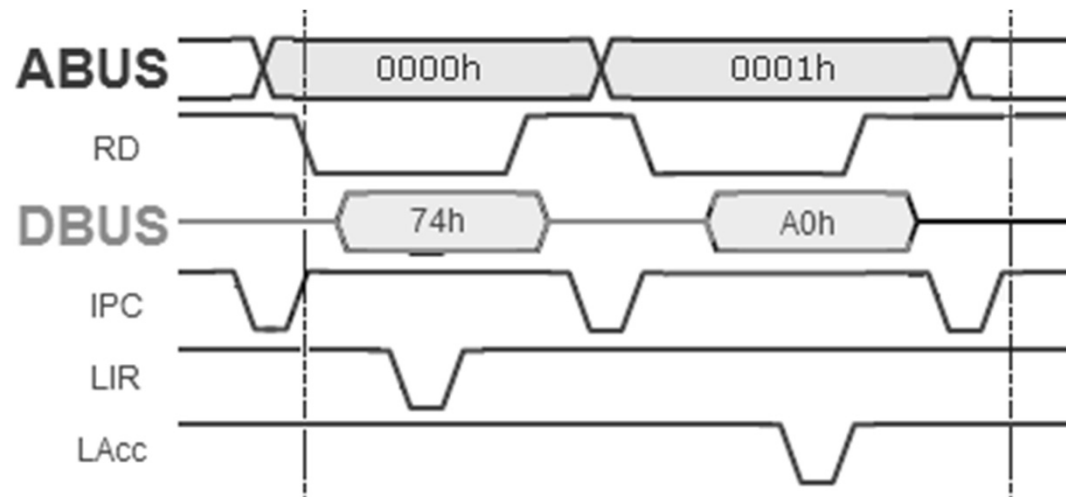
MOV A,#160 74

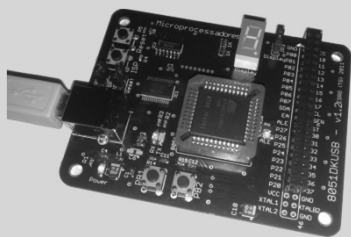
- A instrução será armazenada na memória de código. Há directivas *assembly* (instruções para o assembler) que permitem definir em que endereço pretendemos armazenar a informação;
- Na instrução MOV A,#160 são armazenados dois bytes na memória de programa, o 74h que identifica a instrução “MOV A” e o A0h que é o valor constante que queremos carregar para o Acumulador. Estes dois bytes são armazenados sequencialmente na memória.



Executar um *opcode*

- **Comum a todas as instruções:**
 - Endereçar a instrução a ser executada na memória de código/programa (ABUS=PC);
 - Dar ordem de leitura da instrução à memória de código (activar sinal de RD de CBUS);
 - Guardar o código da instrução, devolvido pela memória de programa, no IR (activar sinal de controlo LIR);
 - Endereçar posição seguinte (incrementar o PC, activando o sinal IPC).
- **Decodificar a instrução e executá-la (depende da instrução) - CTU**
 - Endereçar memória código (ABUS=PC);
 - Ler da memória de programa o valor a carregar em Acc (RD de CBUS);
 - Guardar valor devolvido pela memória em Acc (activar sinal LAcc);
 - Endereçar posição seguinte (activar sinal IPC).





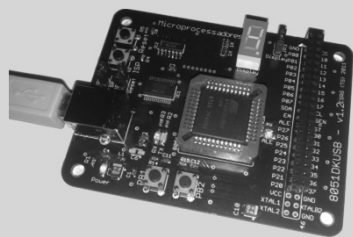
Classificação de processadores

(perspectiva do programador)

Os microprocessadores podem ser classificados usando os seguintes parâmetros:

- Tipos e dimensões dos registos internos
- Organização da memória principal
- A forma como as instruções acedem aos operandos, tanto a partir de registos como a partir de memória

Na realidade, um microprocessador apresenta características de várias classes. A sua classificação baseia-se no comportamento dominante de umas das classes.



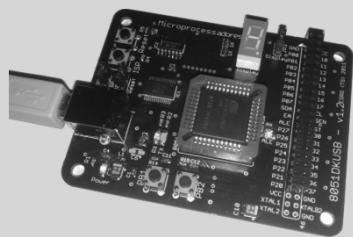
ISA (*Instruction Set Architecture*)

- **Contém informação necessária para interagir com o microprocessador, mas não os detalhes de como o microprocessador é projectado e implementado**
- **É essencialmente uma visão que o programador tem do microprocessador**

Fornece os detalhes que o programador necessita para escrever um programa para um determinado microprocessador, ou os detalhes que um compilador precisa para compilar um programa escrito numa linguagem alto nível

- **Fornece detalhes sobre os registos, acessíveis pelo programador**

O ISA deve especificar quais são estes registos, os seus comprimentos (em bits) e as instruções que podem acedê-los

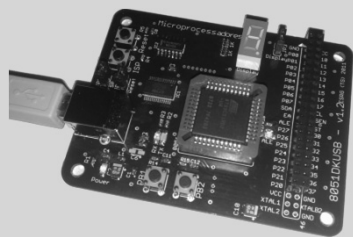


ISA (*Instruction Set Architecture*)

- **Fornece informações necessárias para a interacção com a memória**

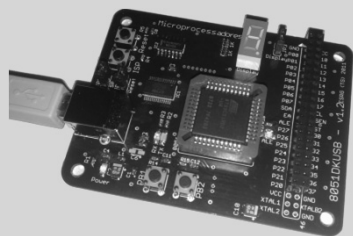
Por exemplo, muitos microprocessadores requerem que as instruções estejam alinhadas (isto é, começam em localizações específicas de memória)

- **Fornece informações sobre a forma como o microprocessador reage às interrupções do ponto de vista do programador**
- ISA fornece instruções para activação/desactivação das interrupções
- Contudo, as acções físicas necessárias para reconhecer a ocorrência de uma interrupção externa não fazem parte do ISA



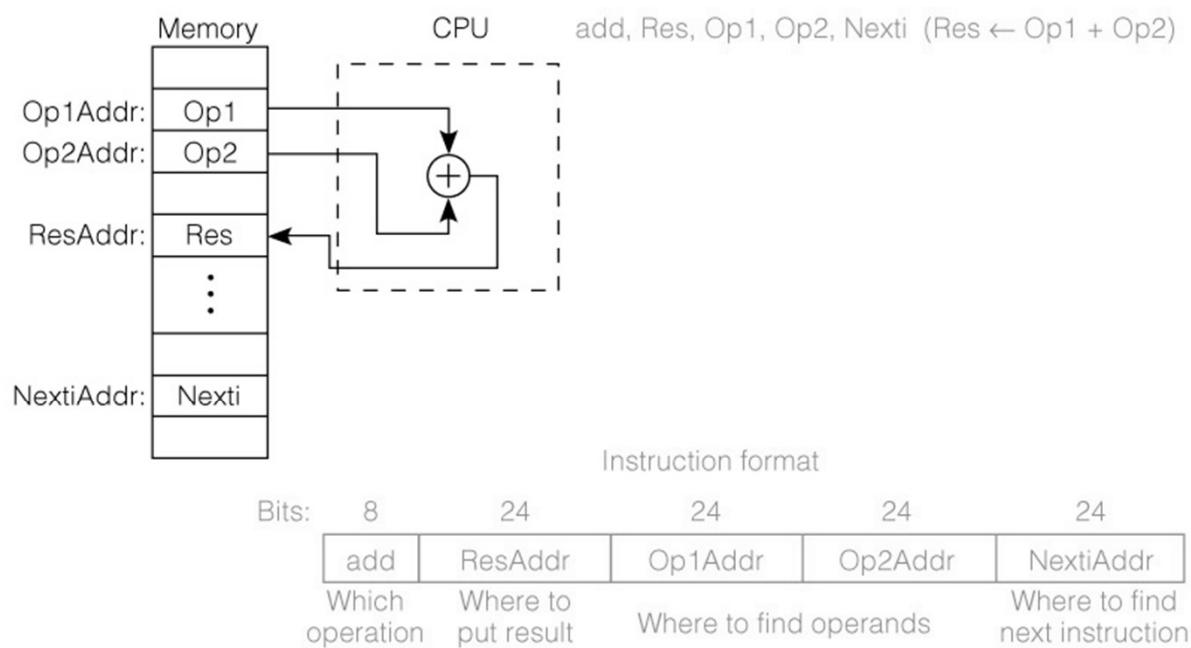
Opcode – Código de operação

- **Que tipo de informação deve uma instrução especificar?**
 - **Operação a ser executada**
 - **Localização dos operandos**
 - **Localização de armazenamento do resultado**
 - **Localização da próxima instrução**

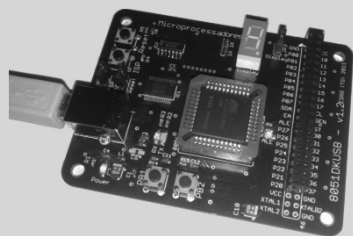


Máquinas de 4-endereços

A instrução especifica de forma explícita os endereços dos operandos, do resultado e da próxima instrução

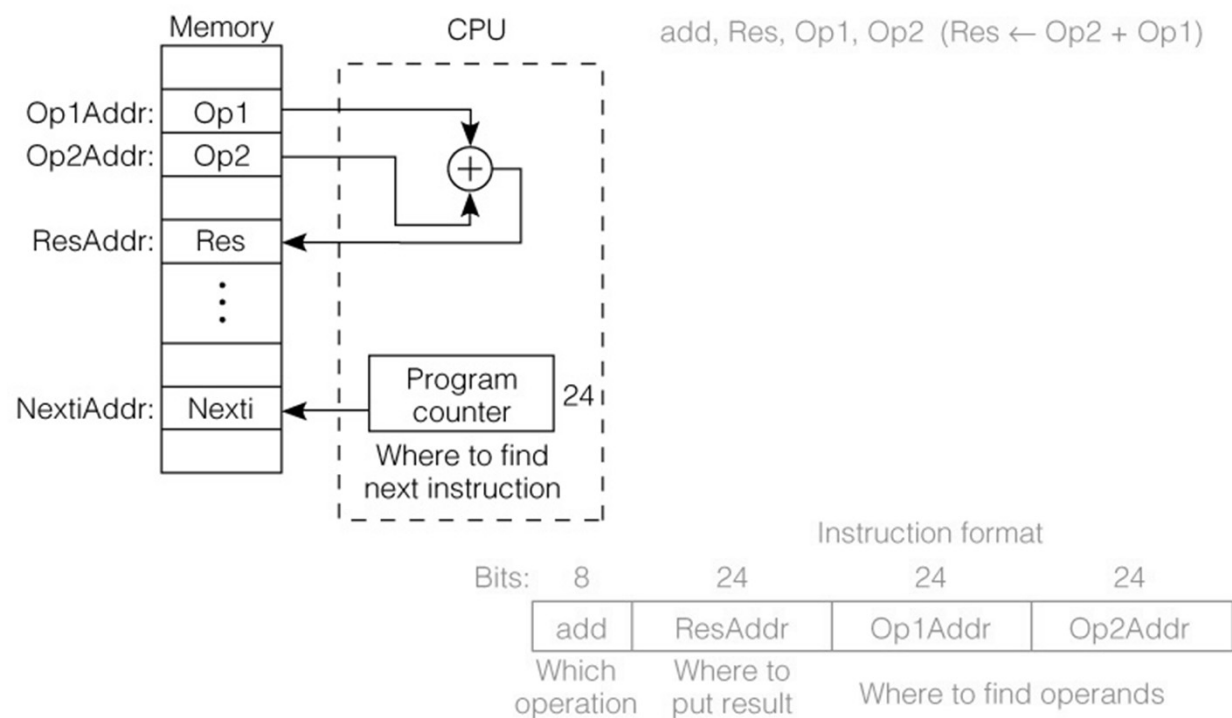


Copyright © 2004 Pearson Prentice Hall, Inc.

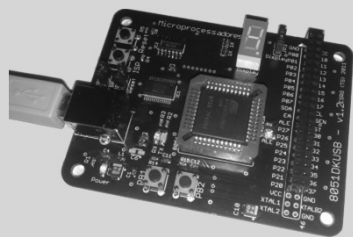


Máquinas de 3-endereços

Endereço da próxima instrução (exceptuando as instruções de salto) é mantida num dos registos de estado do CPU – O *Program Counter* (PC)

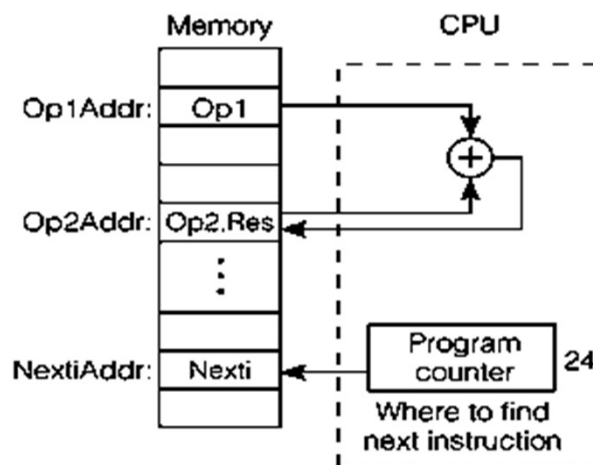


Copyright © 2004 Pearson Prentice Hall, Inc.

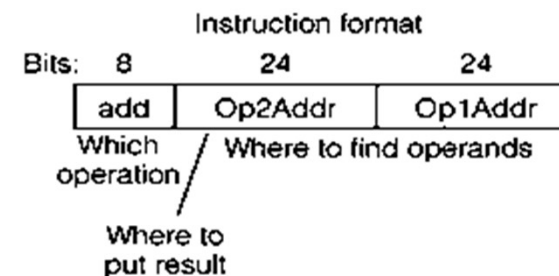


Máquinas de 2-endereços

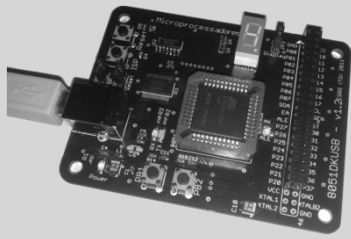
O conteúdo do segundo operando é reescrito com o valor do resultado, isto é, o resultado e o segundo operando partilham o mesmo endereço



add Op2, Op1 ($Op2 \leftarrow Op1 + Op2$)

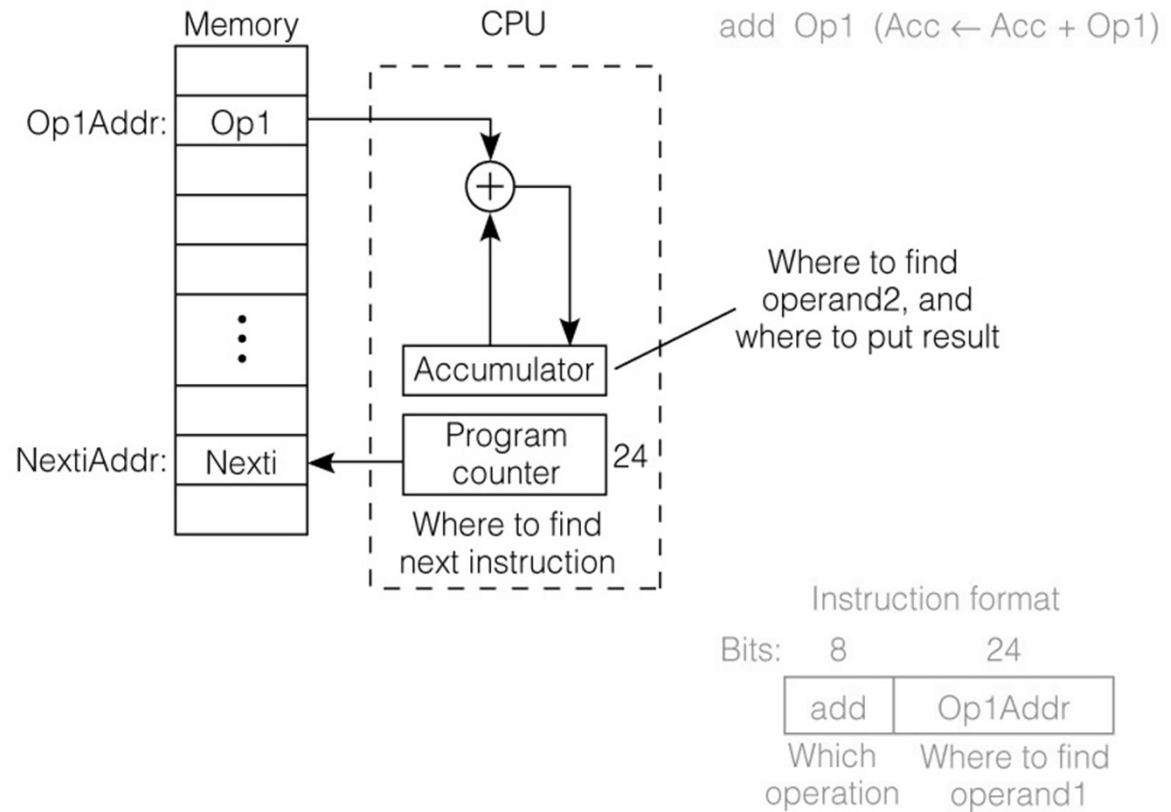


Copyright © 2004 Pearson Prentice Hall, Inc.

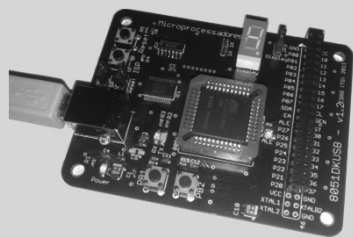


Máquinas de 1-endereço

1. **Torna-se necessário instruções para carregar (*LOAD*) e armazenar (*STORE*) um dos operando**
2. **O acumulador fornece um dos operandos e armazena o resultado**



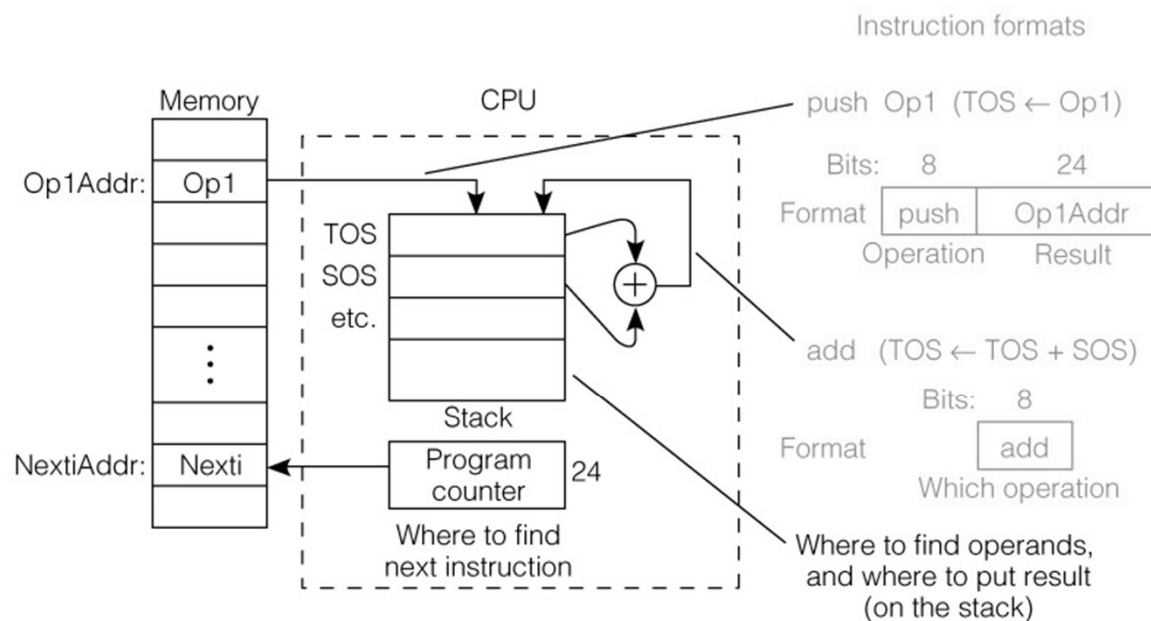
Copyright © 2004 Pearson Prentice Hall, Inc.



Máquinas de 0-endereço

Stack Machine

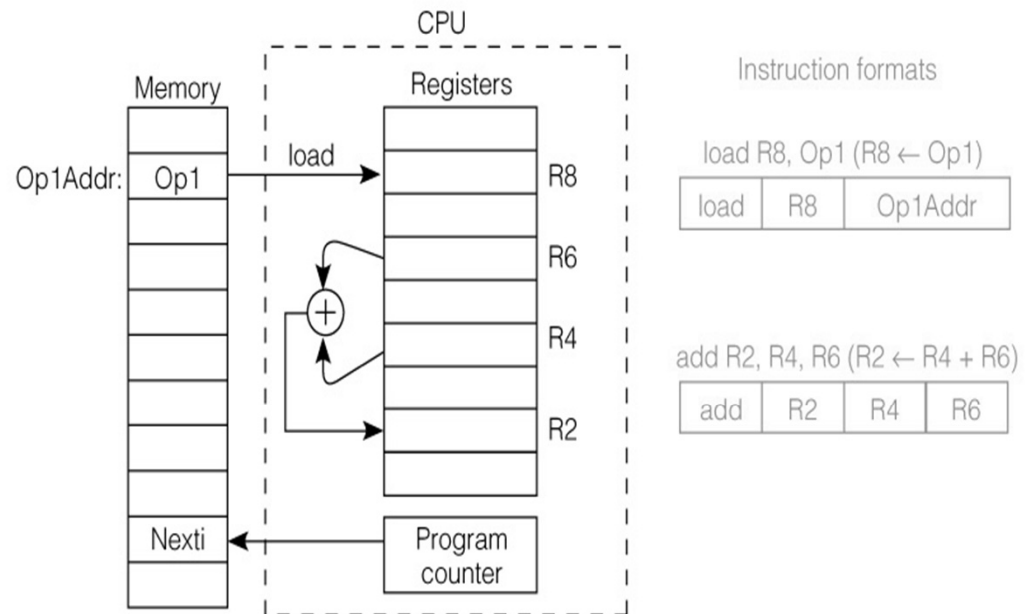
Usa a pilha para ambos os operandos e por isso, será necessário uma instrução para a inserção do operando no topo da pilha (*PUSH*) e outra para a remoção do resultado do topo da pilha (*POP*)



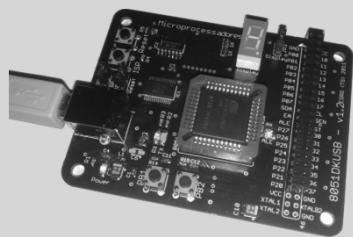
Copyright © 2004 Pearson Prentice Hall, Inc.

Máquinas de Registos genéricos (1-1/2-endereços)

1. Os n registos internos são especificados por meio-endereços usando $\log_2(n)$ bits
2. As instruções de *LOAD* e *STORE* especificam um endereço longo e um meio-endereço: 1-1/2 endereços
3. Operações aritméticas de dois operandos especificam 3 meios-endereços



Copyright © 2004 Pearson Prentice Hall, Inc.



ISA e Níveis de linguagens de programação

1. Linguagens de alto nível (C/C++, Java, Fortran, Pascal, etc)

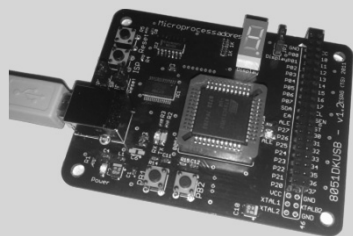
- Independentes da plataforma
- Uma mesma instrução alto-nível pode ser convertido para sequências diferentes de instruções em código máquina

2. Linguagem Assembly (nível de abstracção intermédio)

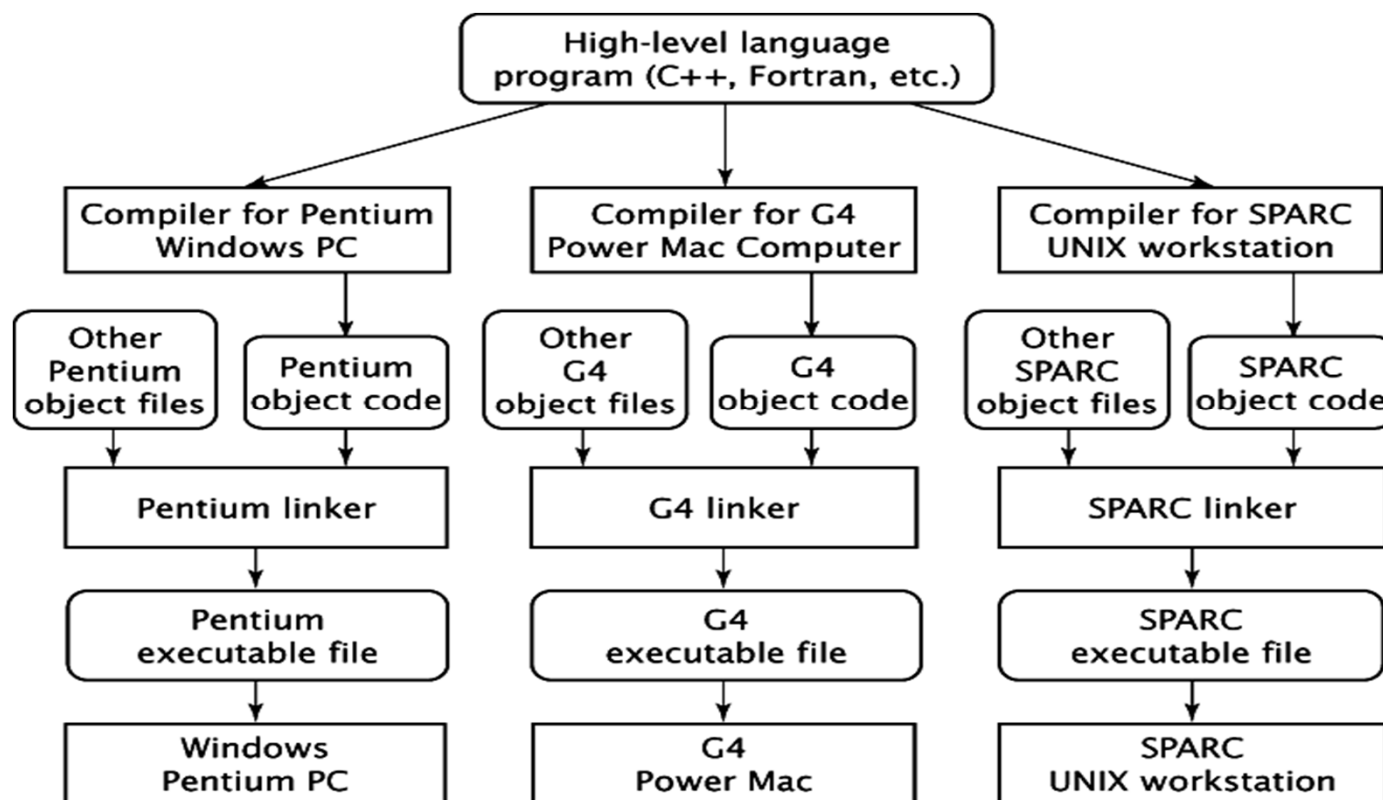
- Cada microprocessador fornece uma linguagem *assembly* própria
- Cada instrução em assembly corresponde a uma única instrução em código máquina
- Normalmente, os fabricantes projectam novos microprocessadores de modo a garantir *backward compatibility* com microprocessadores anteriores

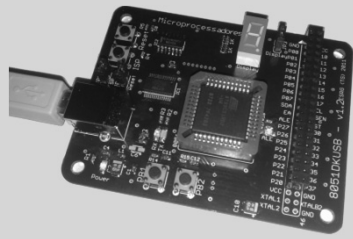
3. Linguagem Máquina (muito pouco abstracta)

- Contém os valores binários que fazem o processador executar determinadas operações

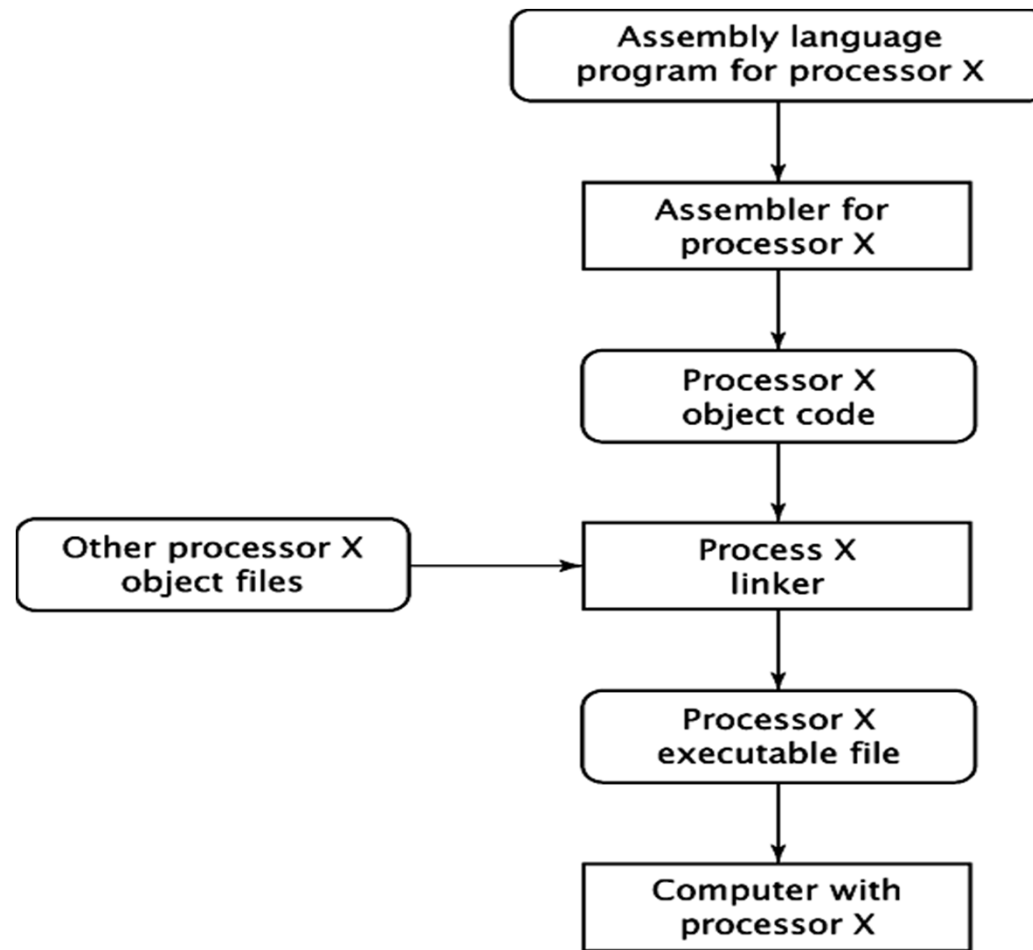


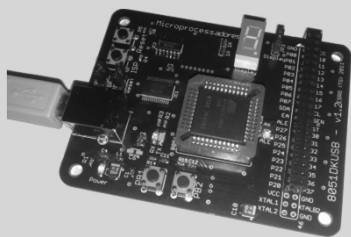
Processo de compilação





Processo de “*assembling*”





Quais os principais atributos das instruções *assembly*?

1. Tipos de instruções

As instruções podem ser agrupadas por classes em função dos tipos de operações que efectuam

2. Tipos de dados

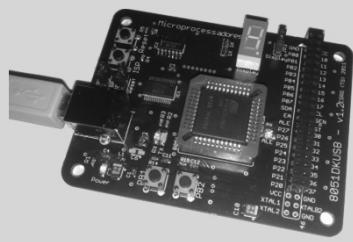
- i. O microprocessador efectua operações em diferentes tipos de dados
- ii. O conjunto de instruções pode incluir diferentes instruções que efectuam a mesma operação sobre diferentes tipos de dados

3. Modos de endereçamento

- i. Ao escrever/ler um dado na/da memória, o microprocessador deve especificar o endereço de memória que precisa aceder
- ii. Uma instrução *assembly* pode usar um dos vários modos de endereçamento para gerar este endereço

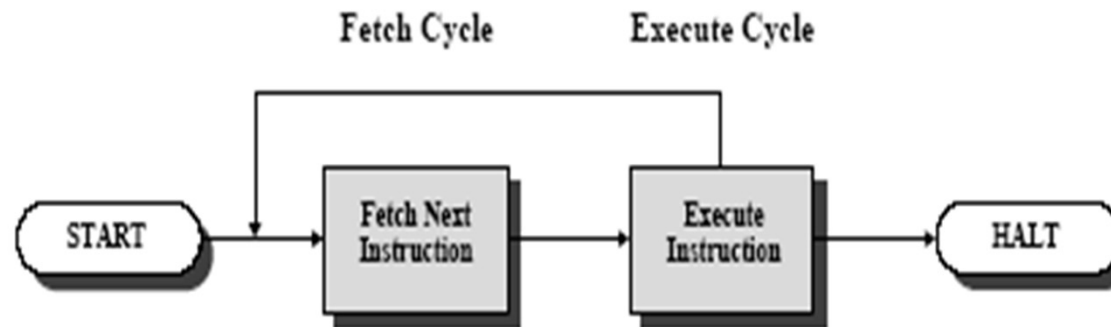
4. Formato das instruções

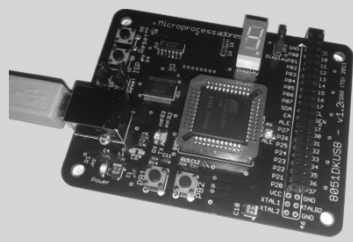
- i. O código da instrução (representação binária da instrução *assembly*) é descrito num determinado formato, em que diferentes grupos de bits representam diferentes partes da instrução
 - i. Um grupo pode representar a operação a ser efectuada (*opcode*), enquanto outros grupos servem para seleccionar os operandos da operação
- ii. Um μ P pode apresentar um único formato para todas as instruções, ou então vários formatos diferentes, tendo cada instrução apenas um formato para o código da instrução



Ciclo de instrução

- Na forma mais simples o processamento de uma instrução consiste nos seguintes passos:
 1. Fetch: Leitura das instruções da memória de programa
 2. Execute: execução de cada instrução



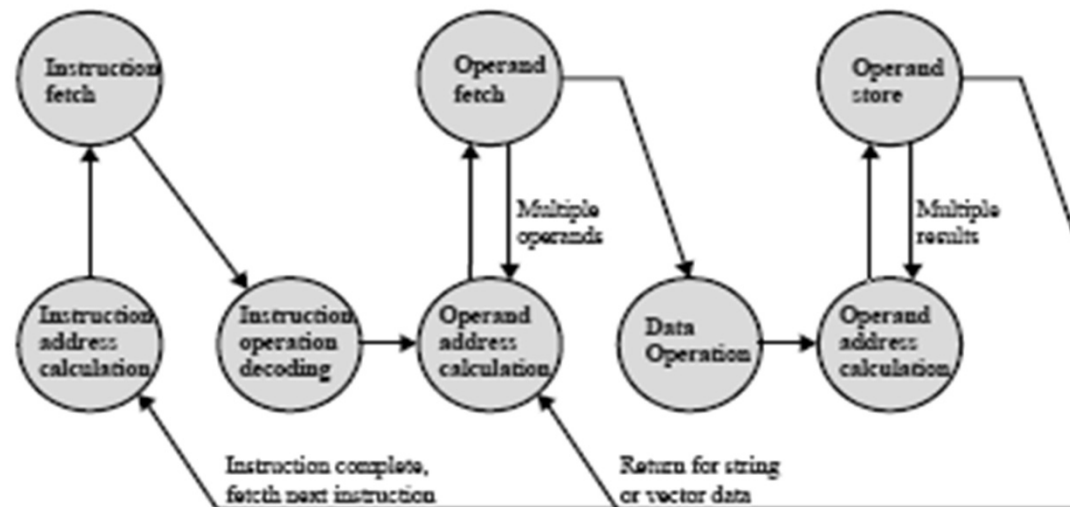


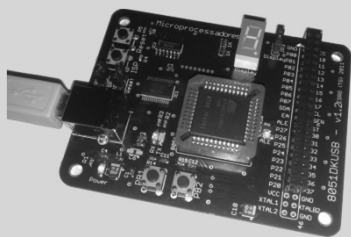
Ciclo de instrução

- Durante a decodificação da instrução, o μP determina qual a instrução de modo a seleccionar a sequência correcta de micro-operações a executar

Repare que cada instrução pode ser composta por uma sequência diferente de micro-operações ao executar

- O ciclo de execução de uma determinada instrução pode envolver várias referências à memória



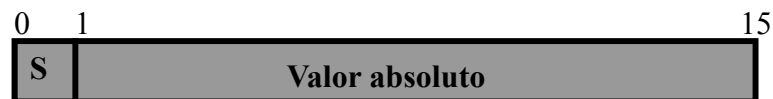


Computador Imaginário

- Formato da Instrução



- Formato para número inteiros



- Registos internos do CPU

- PC (*Program Counter*) = Endereço da instrução
- IR (*Instruction Register*) = Instrução em execução
- AC (*Accumulator*) = Armazenamento temporário

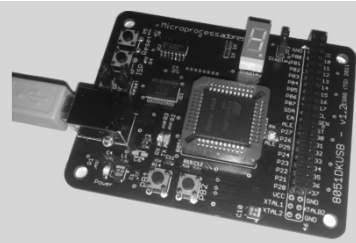
- Lista parcial de *opcodes*

- 0001 = carregar para AC o conteúdo de uma célula de memória
- 0010 = Armazenar conteúdo de AC na memória
- 0101 = adicionar conteúdo de AC com o conteúdo de uma célula de memória

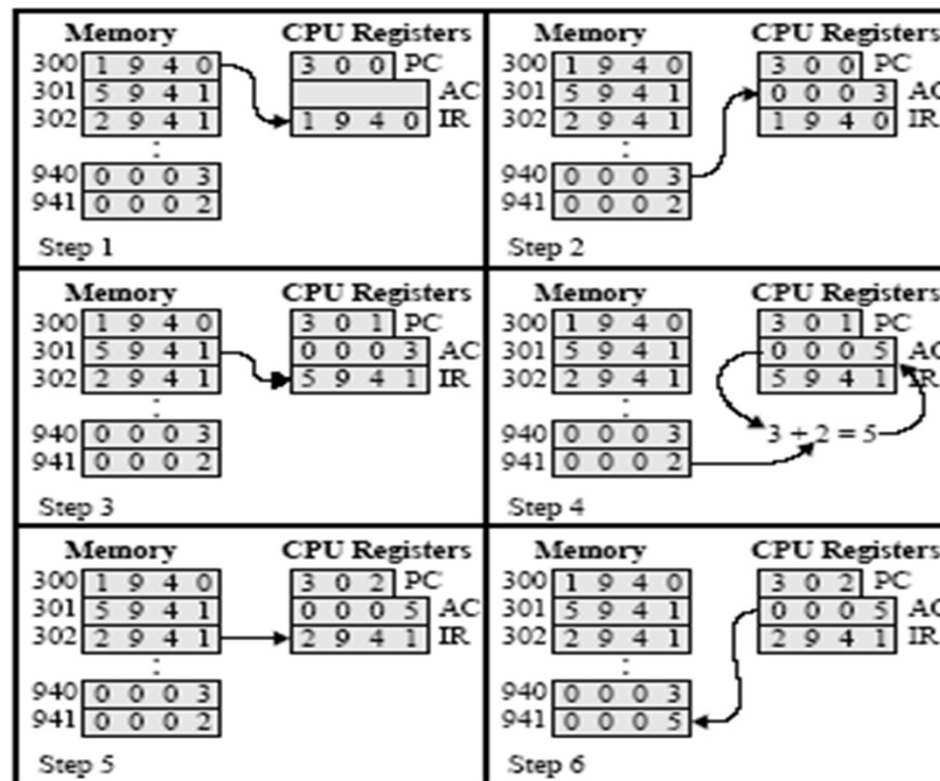
Como organizava a memória?

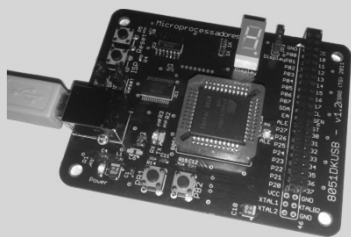
Quantas células de memória podem ser directamente endereçáveis?

Quantos *opcodes* diferentes pode ter o ISA?



- Execução parcial de um programa sobre o computador imaginário





Formato da instrução

Programa *assembly* e código máquina para:
calcular $A = B + C$

4 bits	2 bits	2 bits	2 bits
opcode	operand #1	operand #2	operand #3

ADD A,B,C ($A=B+C$) 1010 00 01 10

(a)

4 bits	2 bits	2 bits
opcode	operand #1	operand #2

MOVE A,B ($A=B$) 1000 00 01
ADD A,C ($A=A+C$) 1010 00 10

(b)

4 bits	2 bits
opcode	operand

LOAD B ($Acc=B$) 0000 01
ADD C ($Acc=Acc+C$) 1010 10
STORE A ($A=Acc$) 0001 00

(c)

4 bits
opcode

PUSH B ($Stack=B$) 0101
PUSH C ($Stack=C,B$) 0110
ADD ($Stack=B+C$) 1010
POP A ($A=stack$) 1100

(d)

Quanto menor for o número de operandos:

1. mais instruções serão necessárias para a realização da mesma tarefa
2. o *hardware* para a implementação de um μP torna-se menos complexo
3. as instruções são executadas mais rapidamente