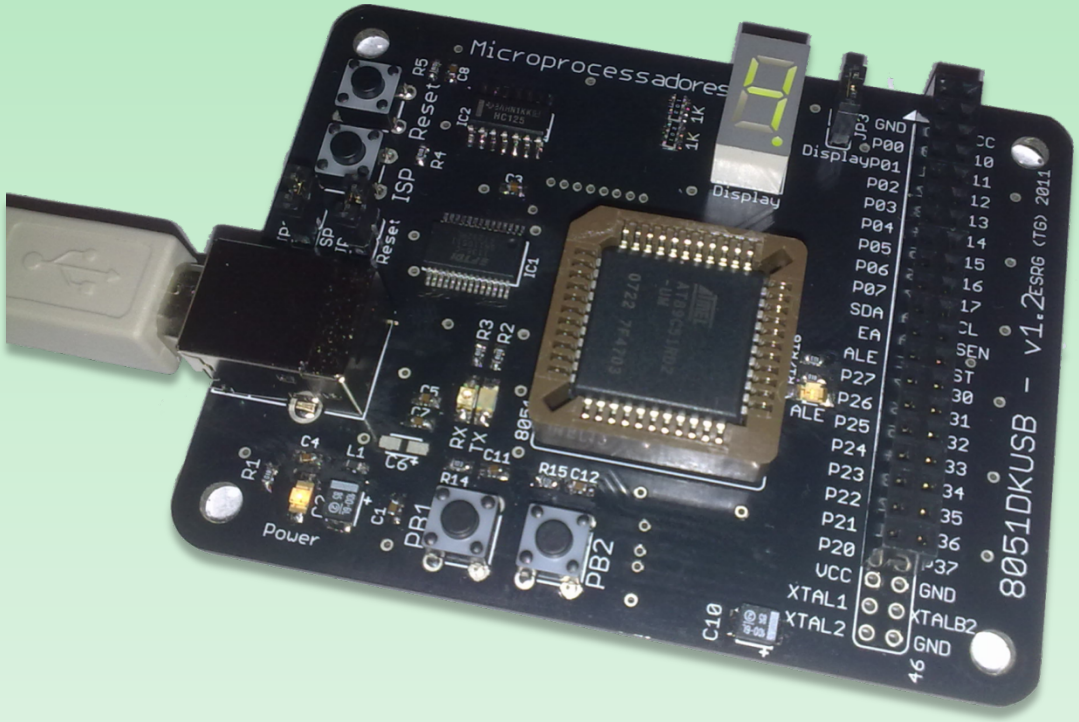
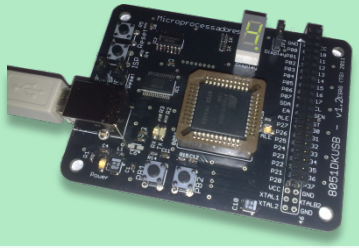
Mestrado Integrado em Eng. Electrónica Industrial e Computadores



Sistema Computadorizado

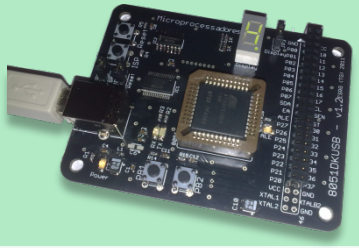
Microcontroladores

2º Ano – A03

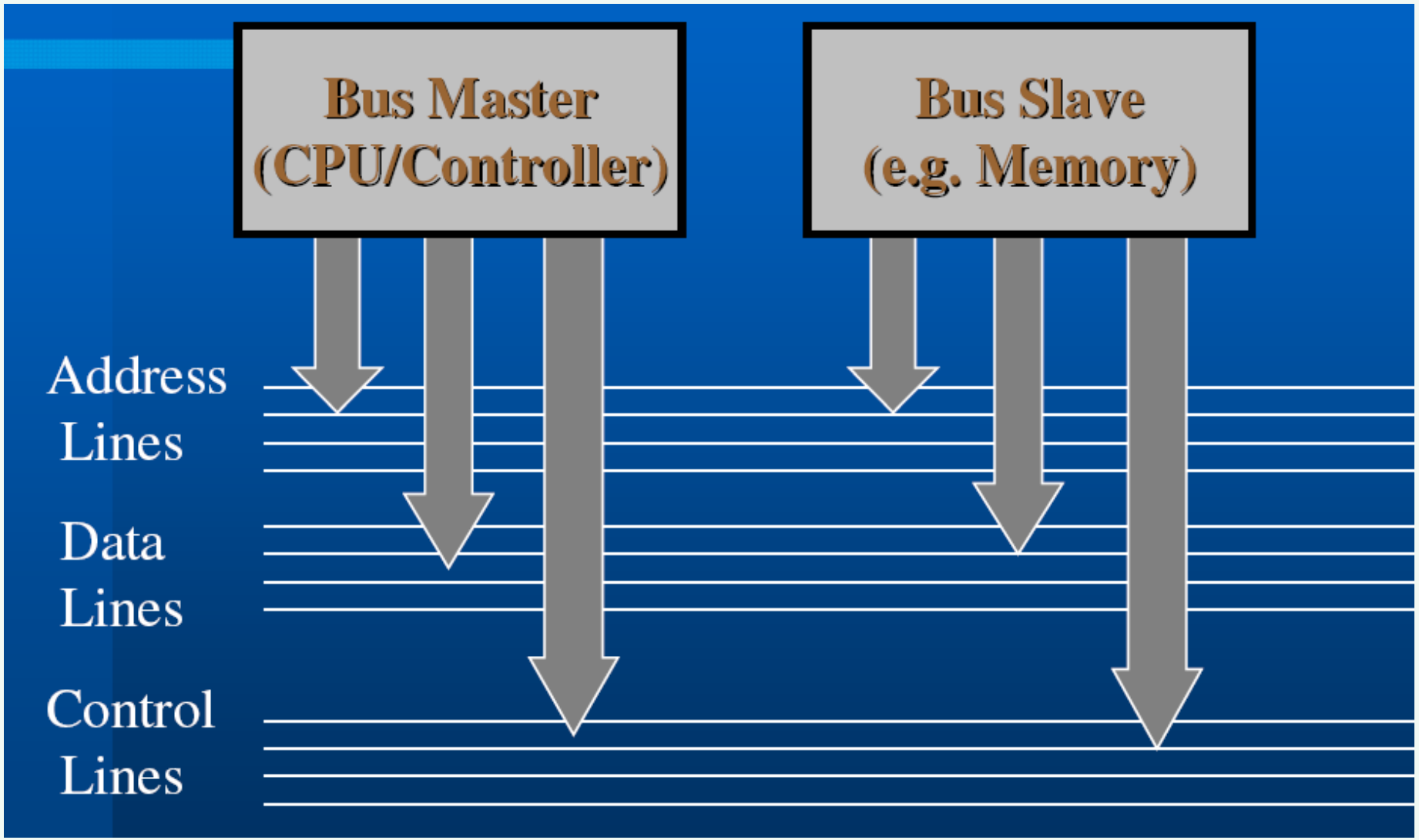


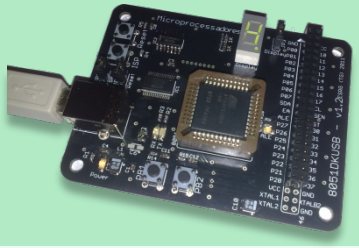
Barramentos

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



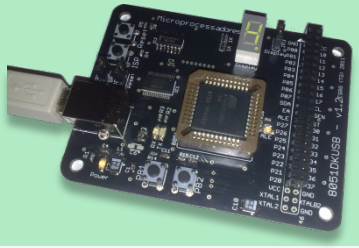
Barramentos





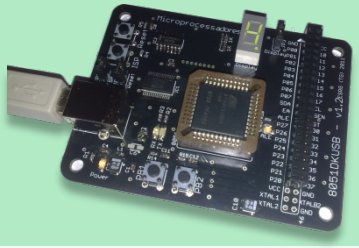
Sequência de Escrita (Write)

- 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).



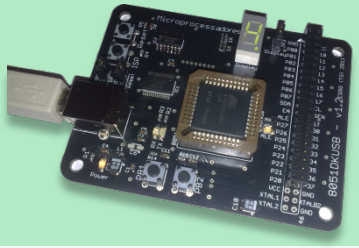
Sequência de leitura (Read)

- 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:

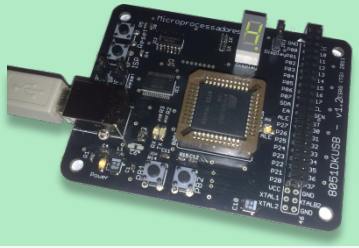
```
int max(int a, int b)
{
   if (a > b) return a;
   else return b;
}
```

  - Mnemónicas das Instruções

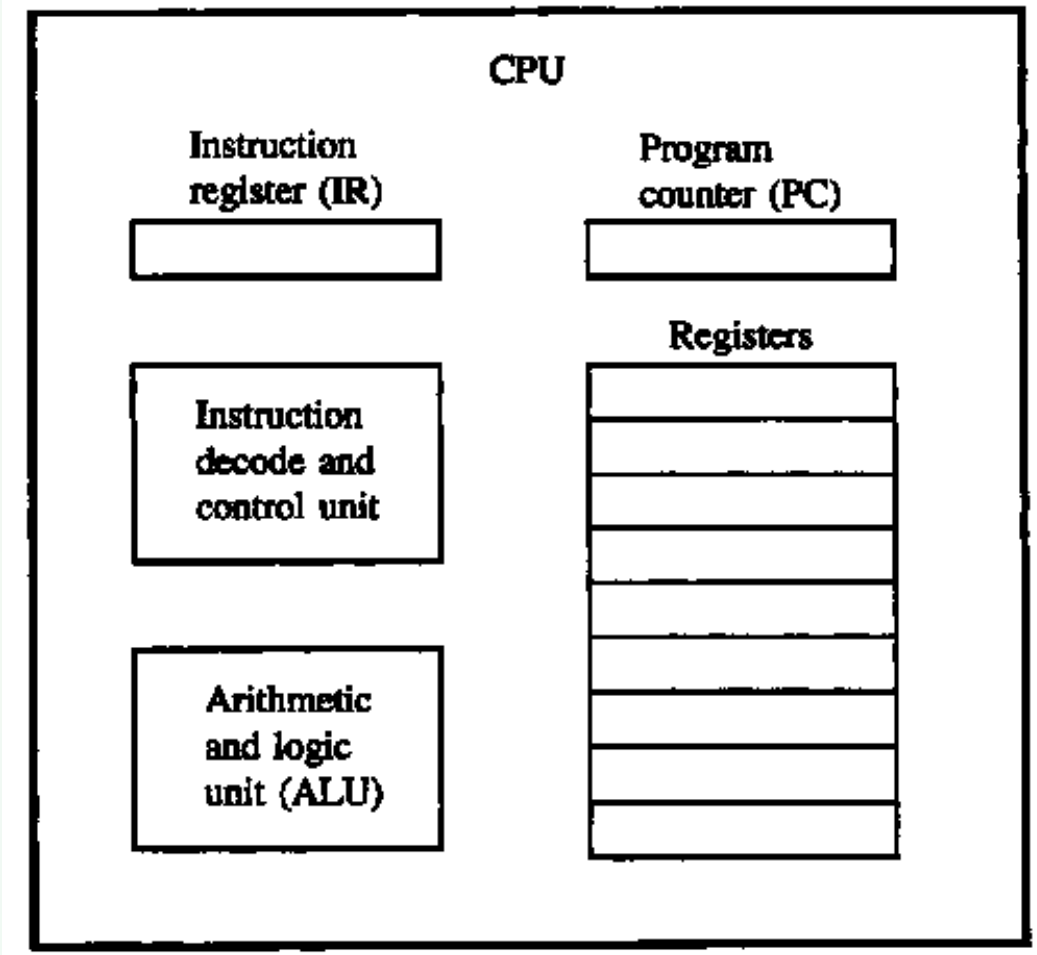
```
        MOV R0,A
        MOV R1,B
        CLR C
        SUBB A,R1
        JC LBL1
        MOV A,R0
        MOV R3,A
        RET
LBL1:   MOV A,R1
        MOV R3,A
        RET
```

  - Código máquina
    90A7CB89210376

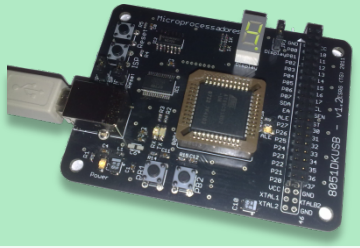
Programa armazenado na memória (RAM / ROM)



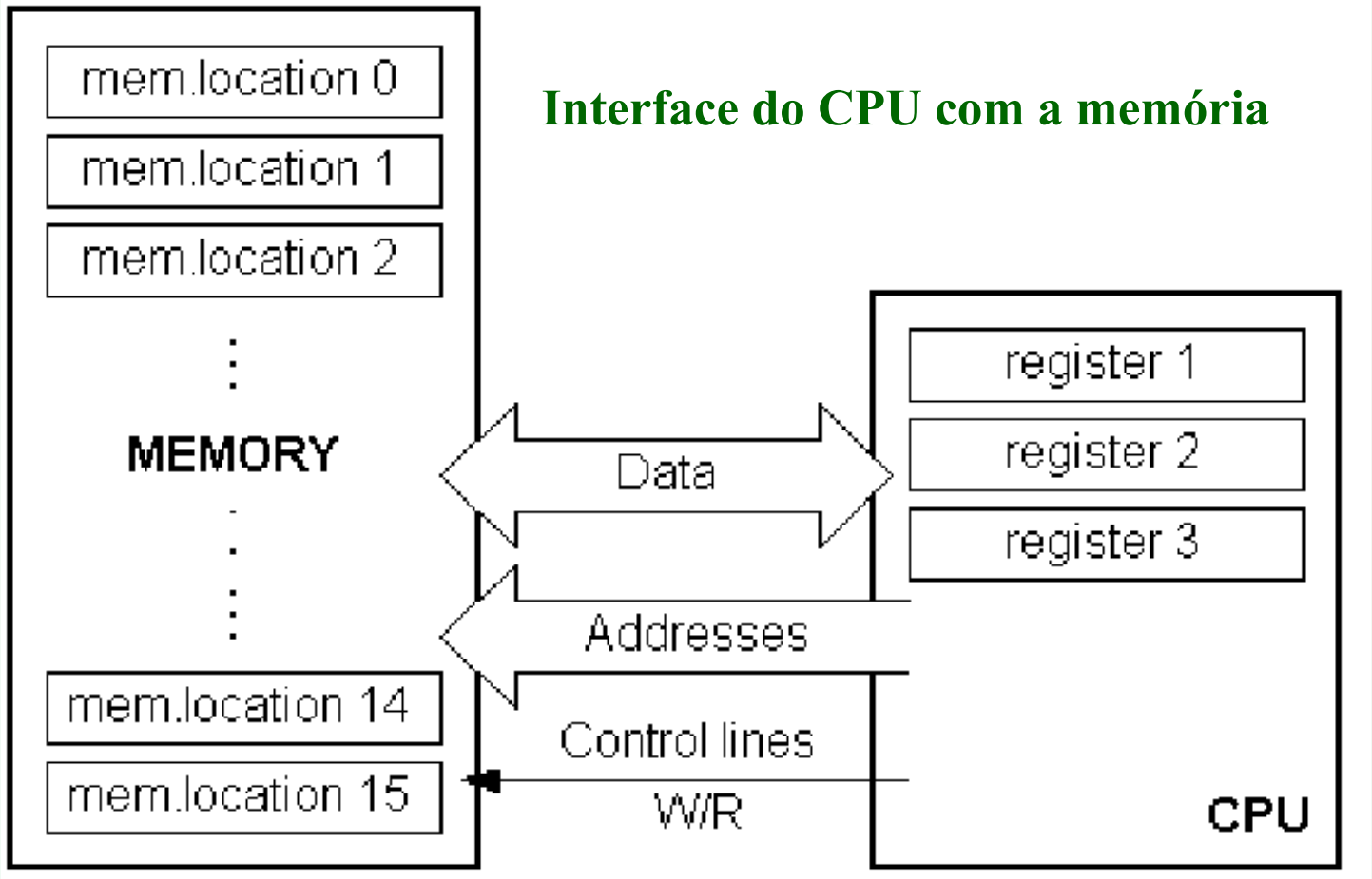
CPU



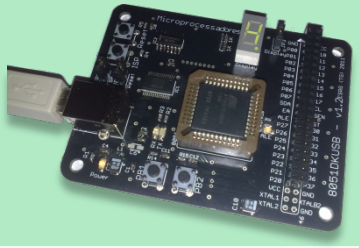
Exemplo de uma unidade de processamento central



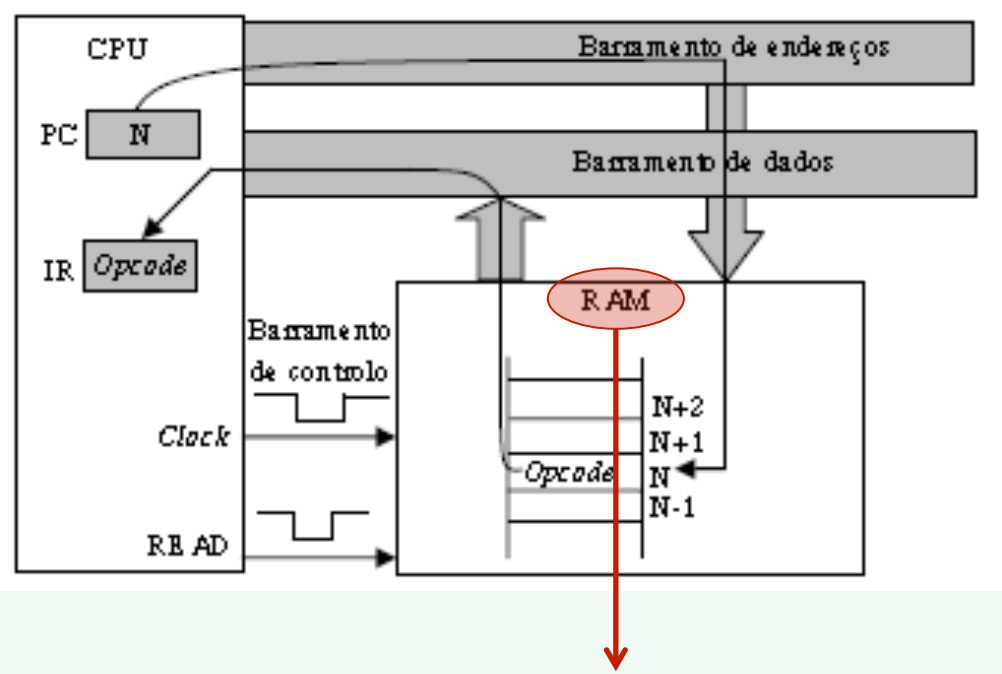
Bus - barramento



Interface do CPU com a memória



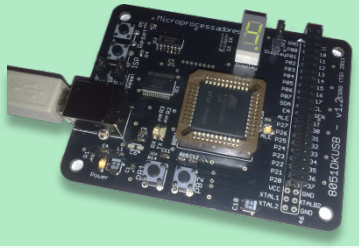
Fetch de um opcode



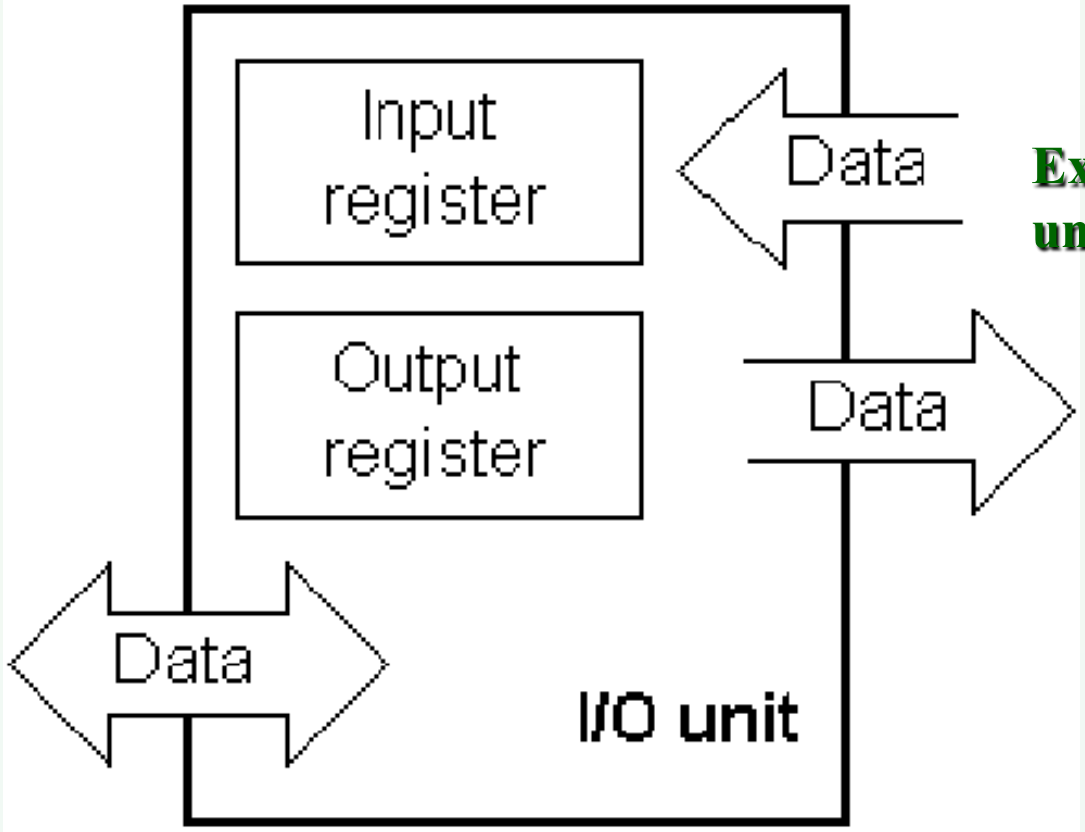
Passos

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;

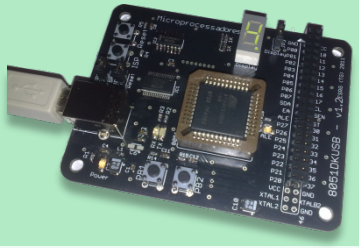
No caso de um microcontrolador esta memória é de que tipo?



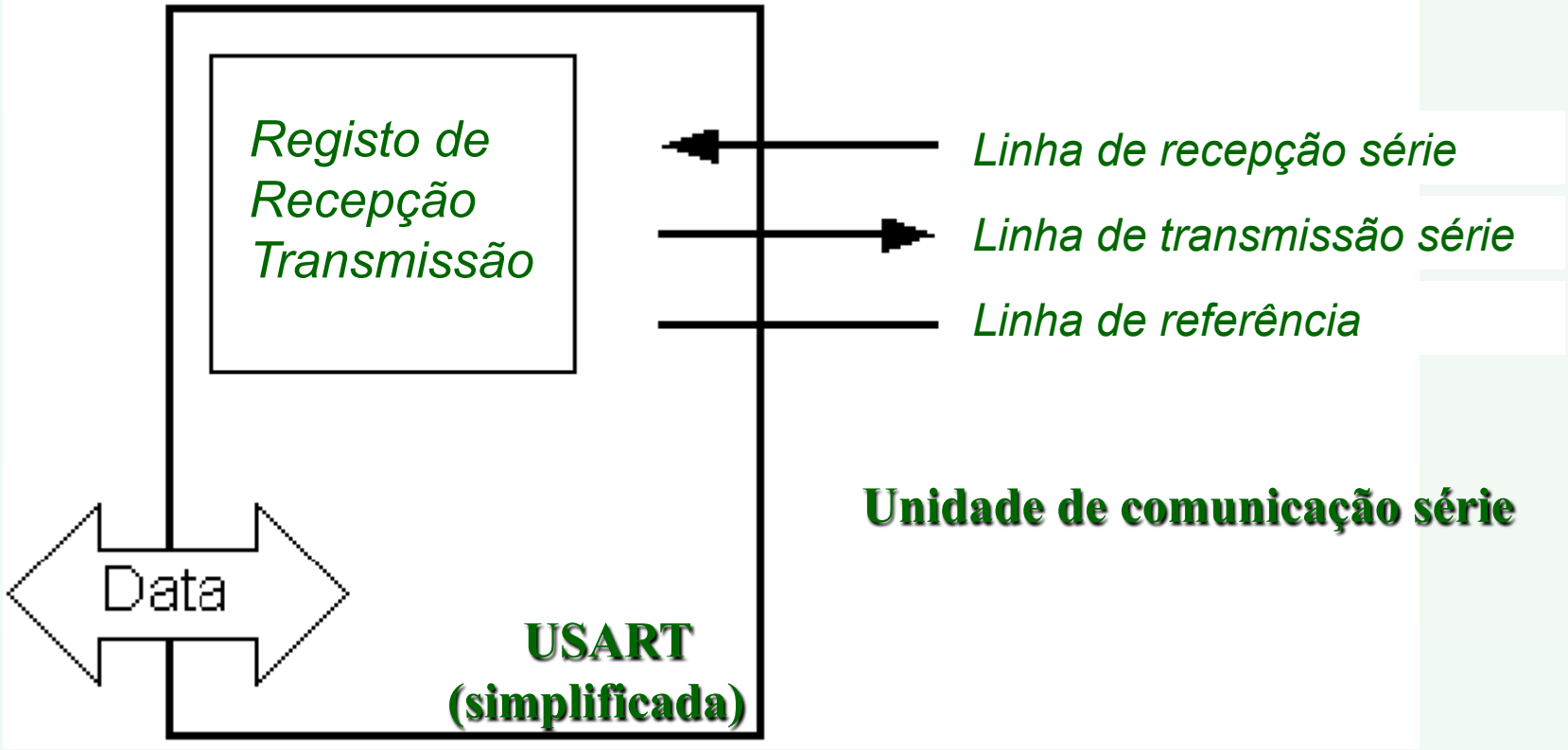
Unidade de Input/Output

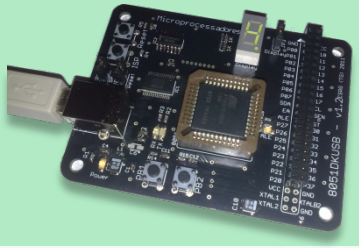


Exemplo simplificado da unidade de I/O (entrada/saída)

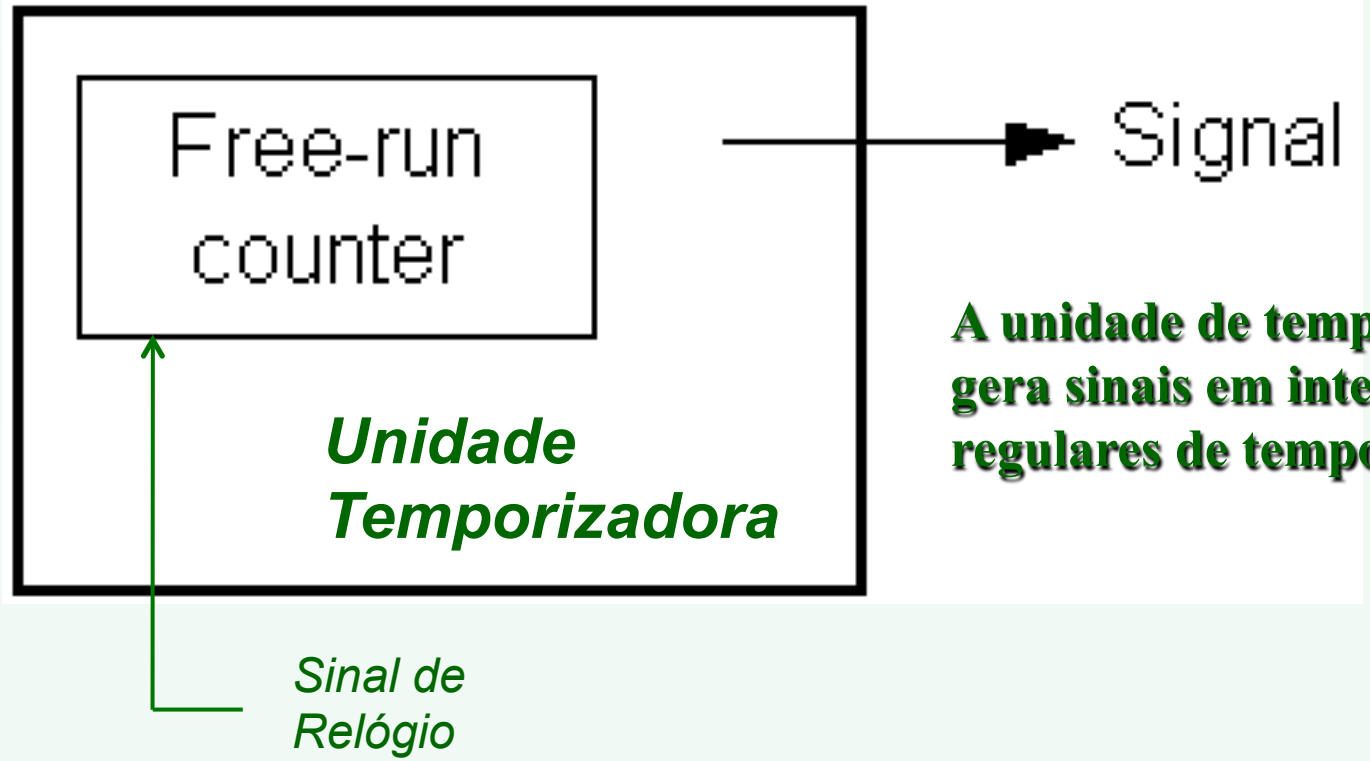


Unidade de Comunicação

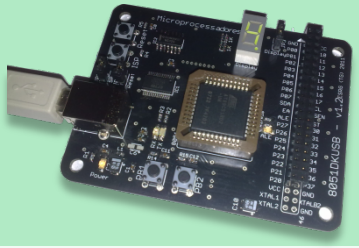




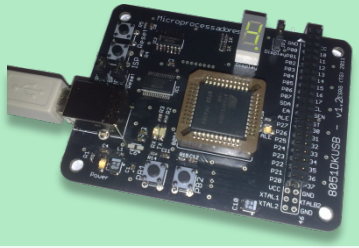
Unidade de temporização



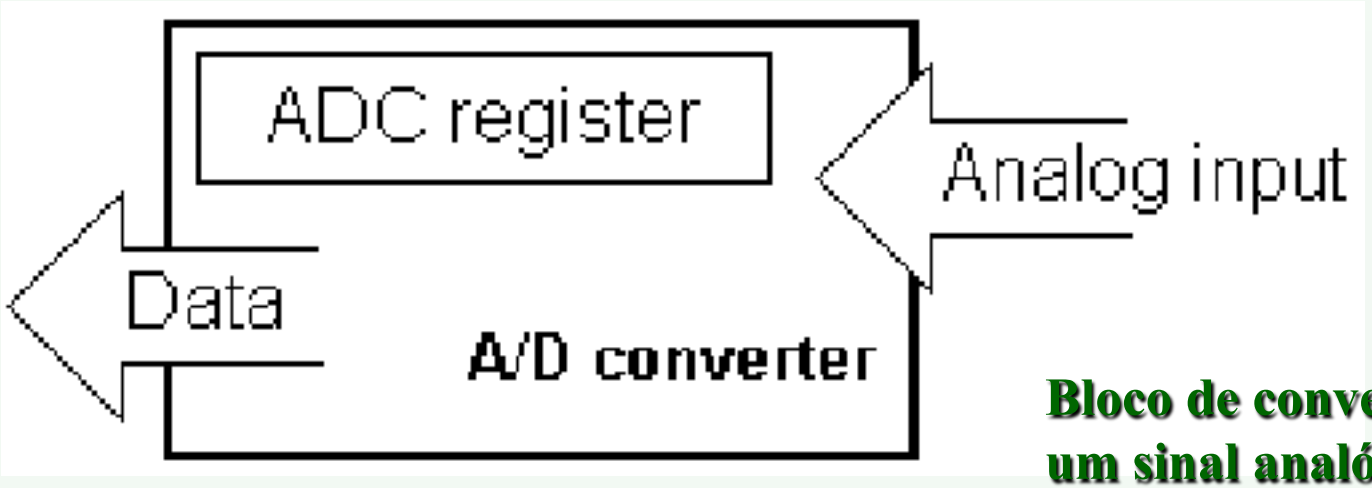
A unidade de temporização gera sinais em intervalos regulares de tempo



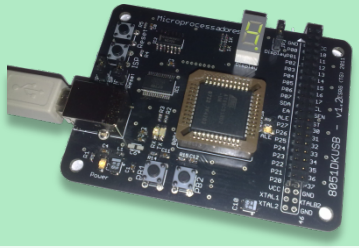
Unidade de Watchdog



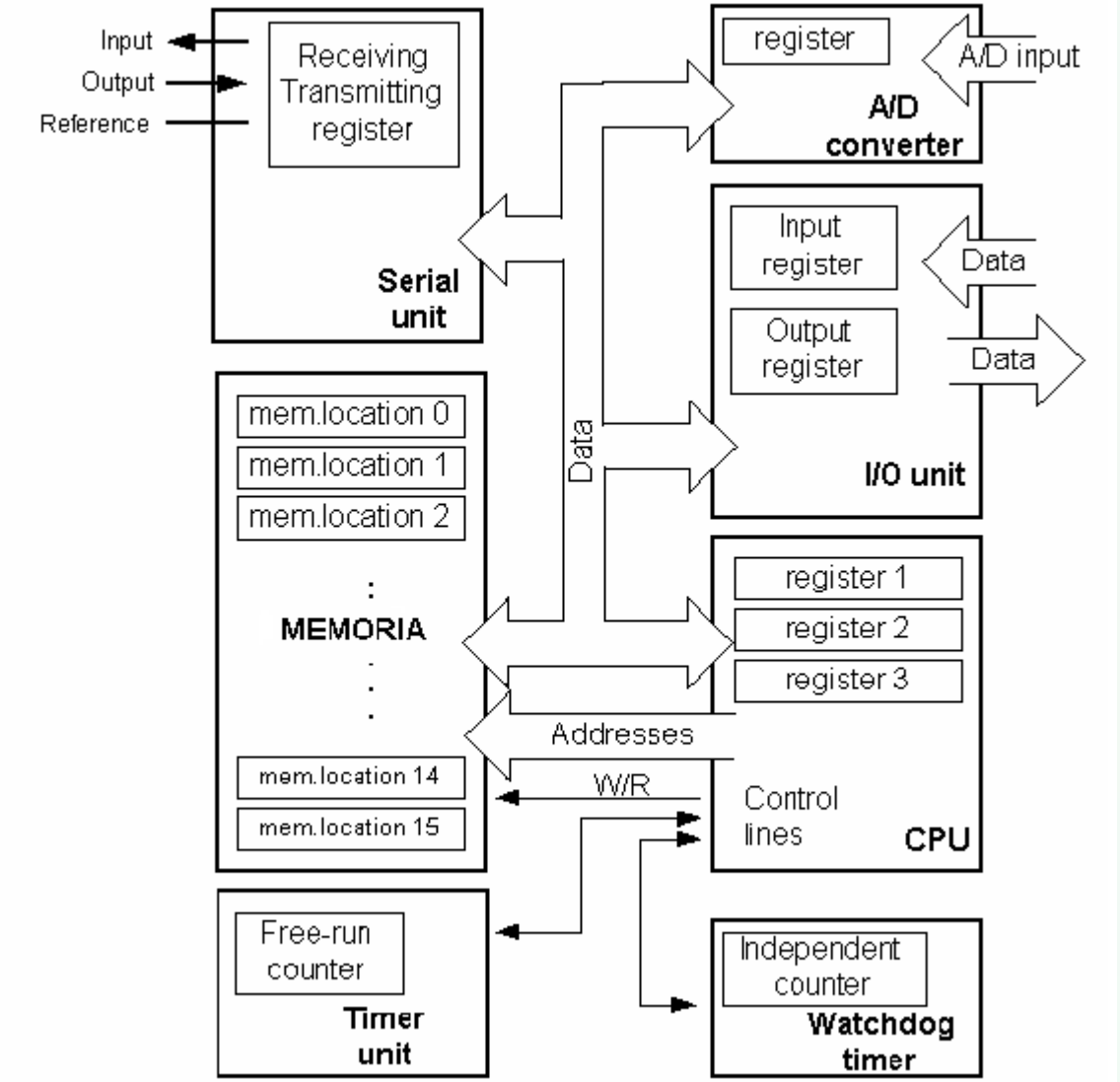
Unidade de Conversão A/D

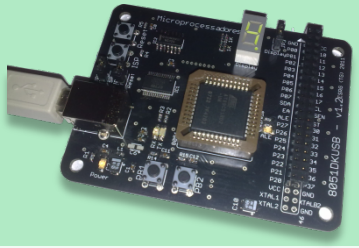


Bloco de conversão de um sinal analógico numa representação digital

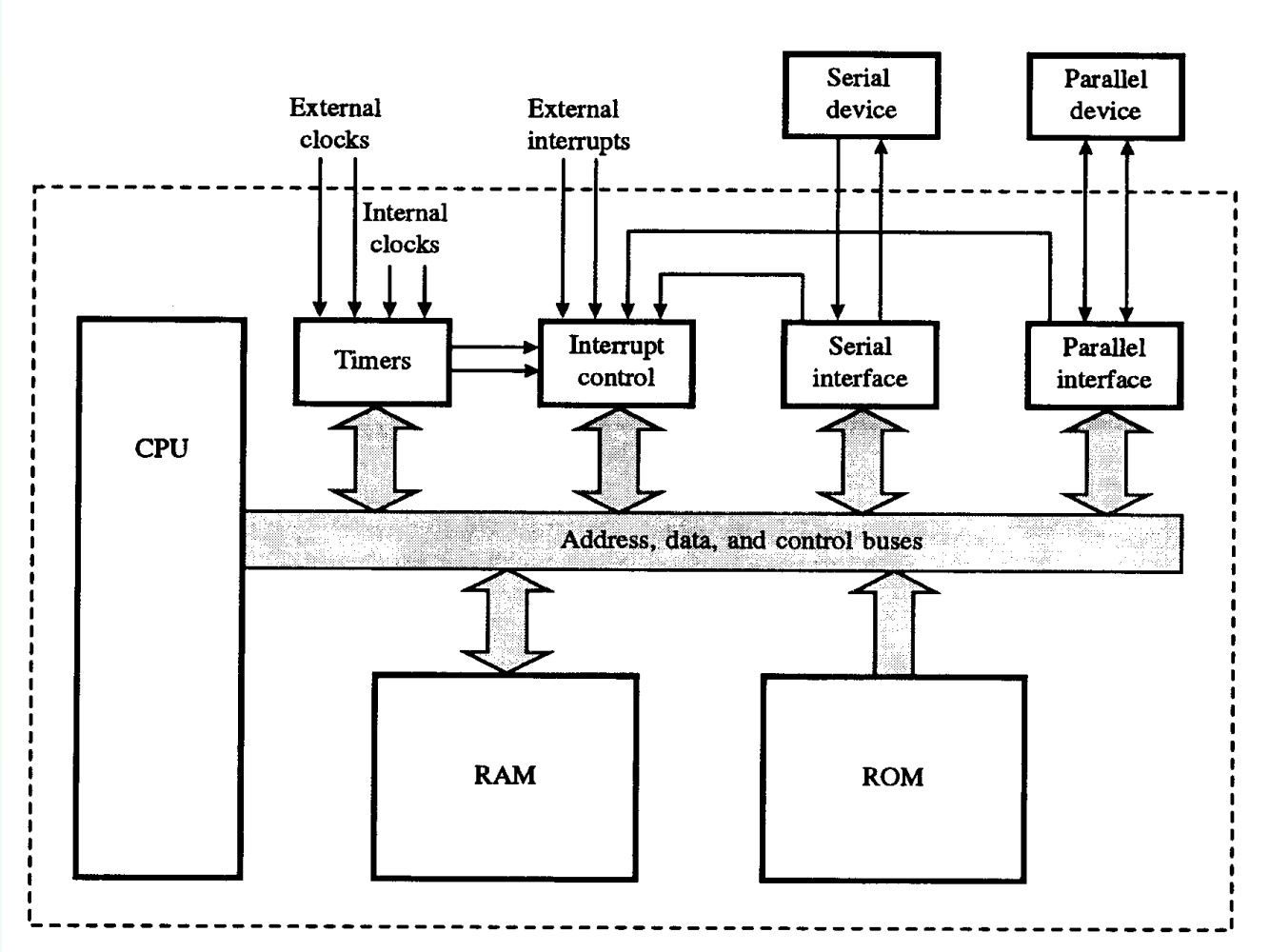


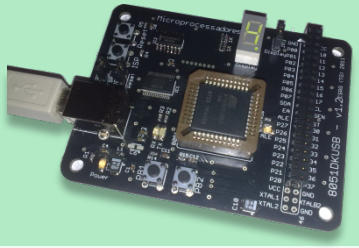
Estrutura de um microcontrolador





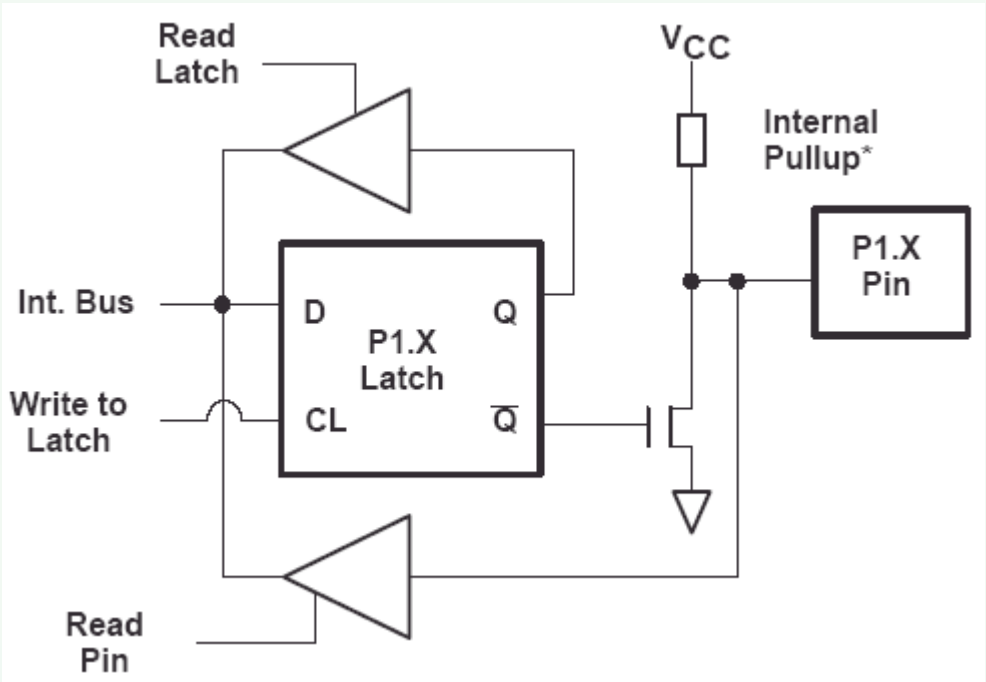
Estrutura genérica

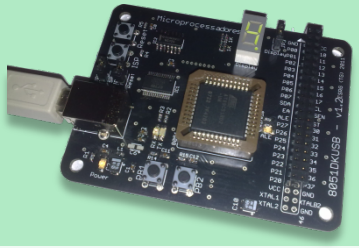




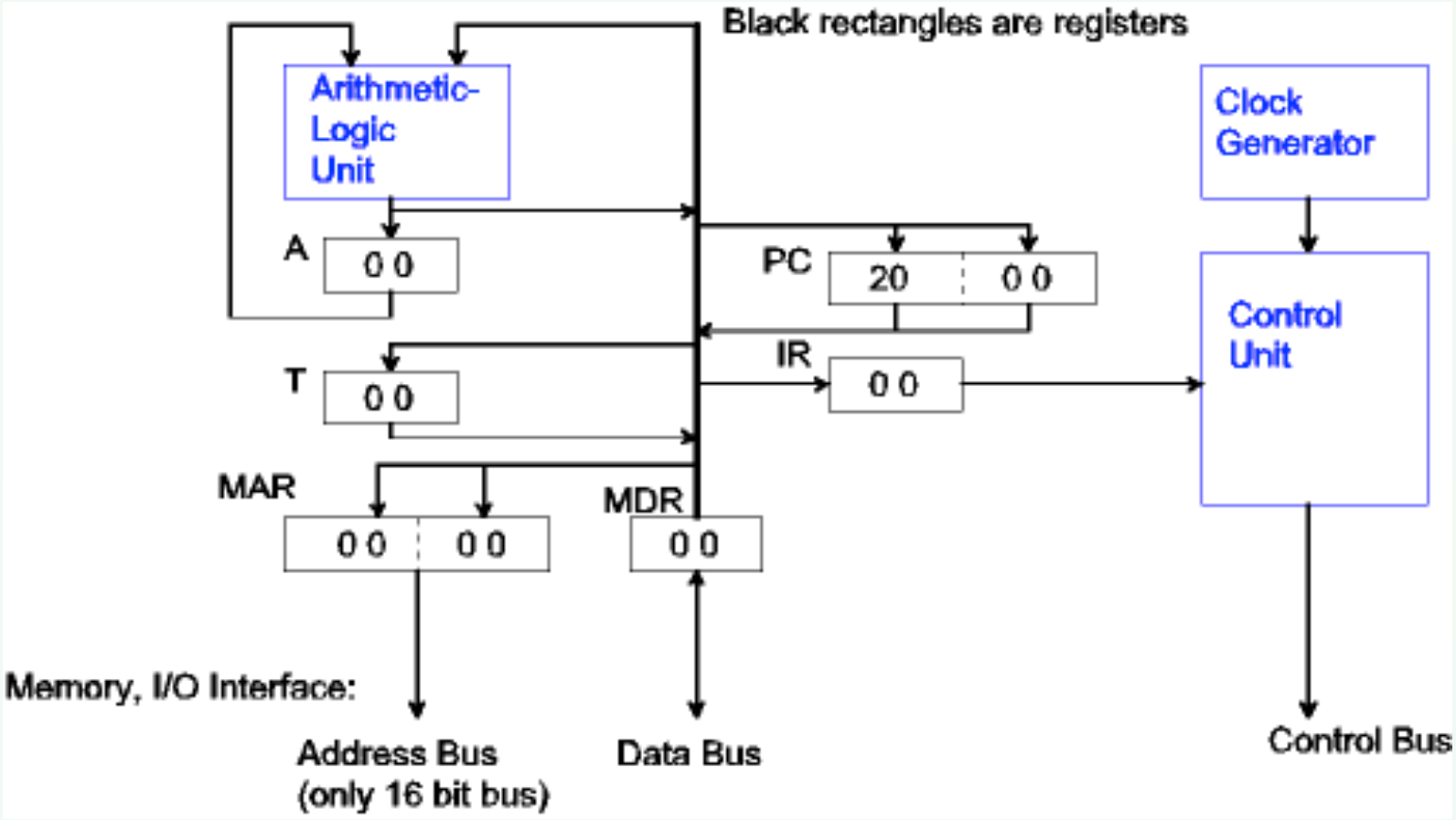
Portos de Entrada/Saída (I/O)

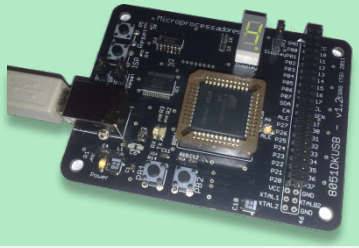
- A saída de um flip-flop (Latch) pode ser ligada directamente aos pinos de um microcontrolador, tornando possível a leitura do valor digital presente no pino ou a escrita de um valor no pino.
- Há registos (8 Latches) que estão directamente conectados aos pinos de um micro. São designados Portos de 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.



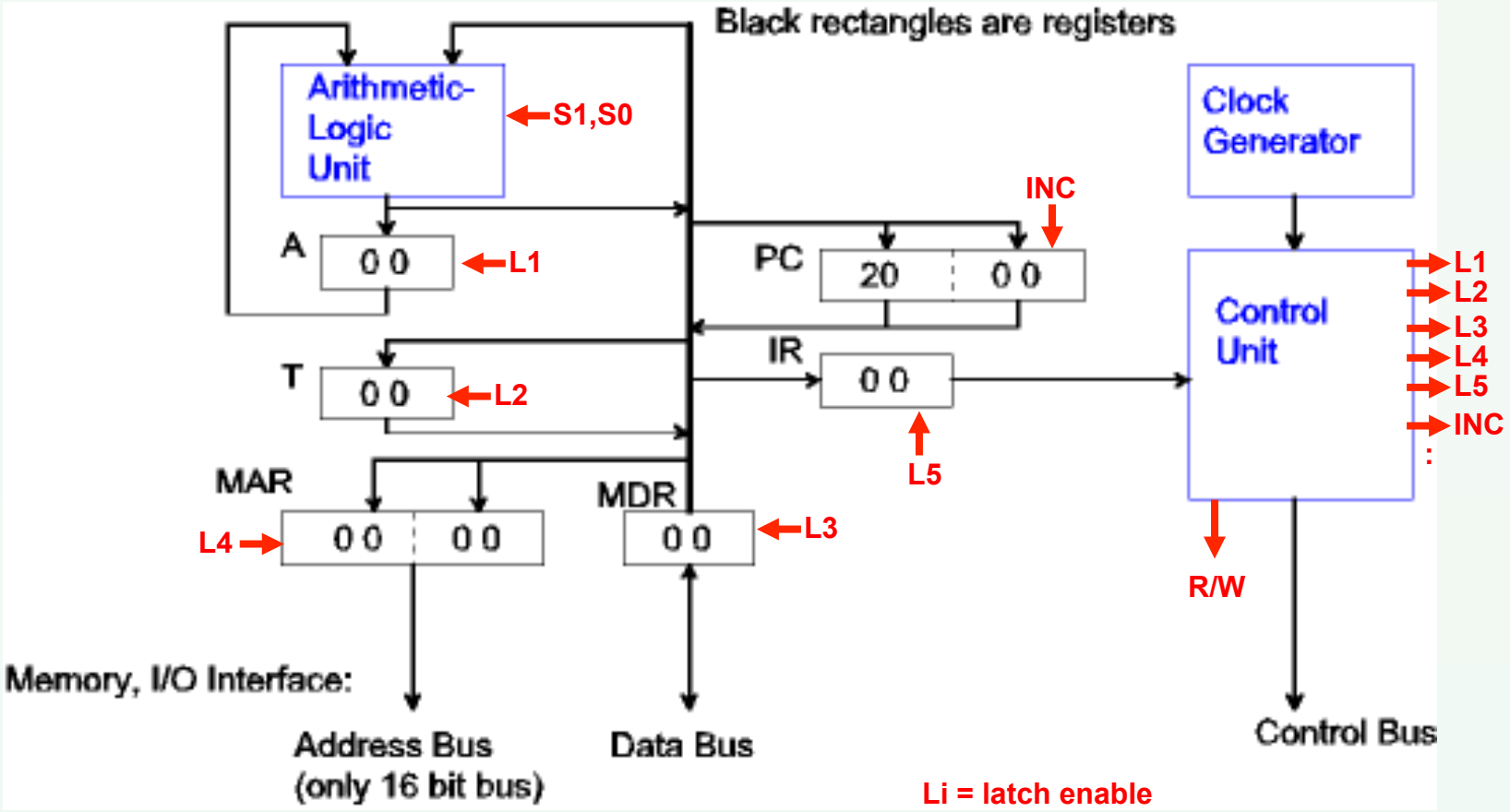


Exemplo de um processador

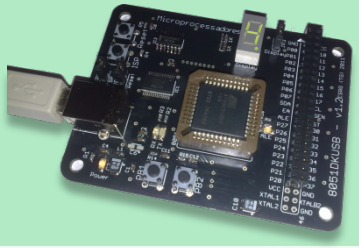




Sinais de controlo



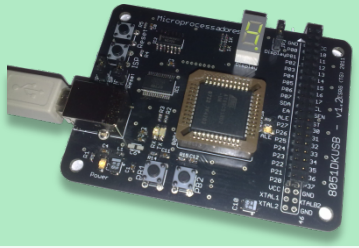
Li = latch enable
INC = increment
R/W = read/write memory
S1,S0 = select ALU function



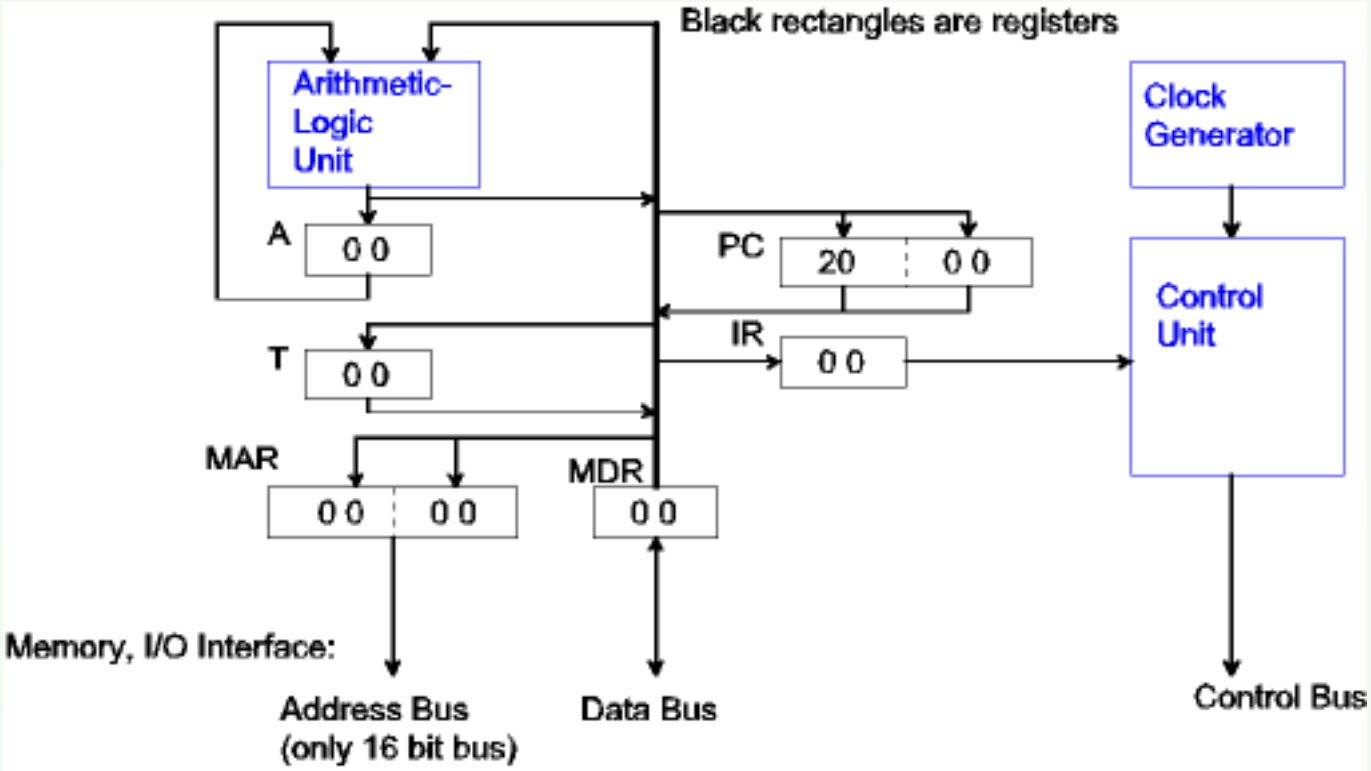
Conteúdo de uma memória

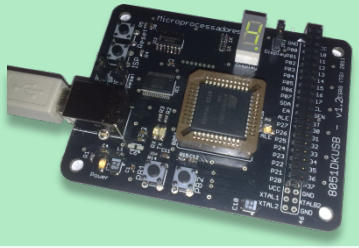
Location	Contents	Instruction
$2000	$B6	LDAA $1000
$2001	$10	
$2002	$00	
$2003	$BB	ADDA $1001
$2004	$10	
$2005	$01	
$2006	$43	DECA
$2007	$7A	STAA $1002
$2008	$10	
$2009	$02	

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



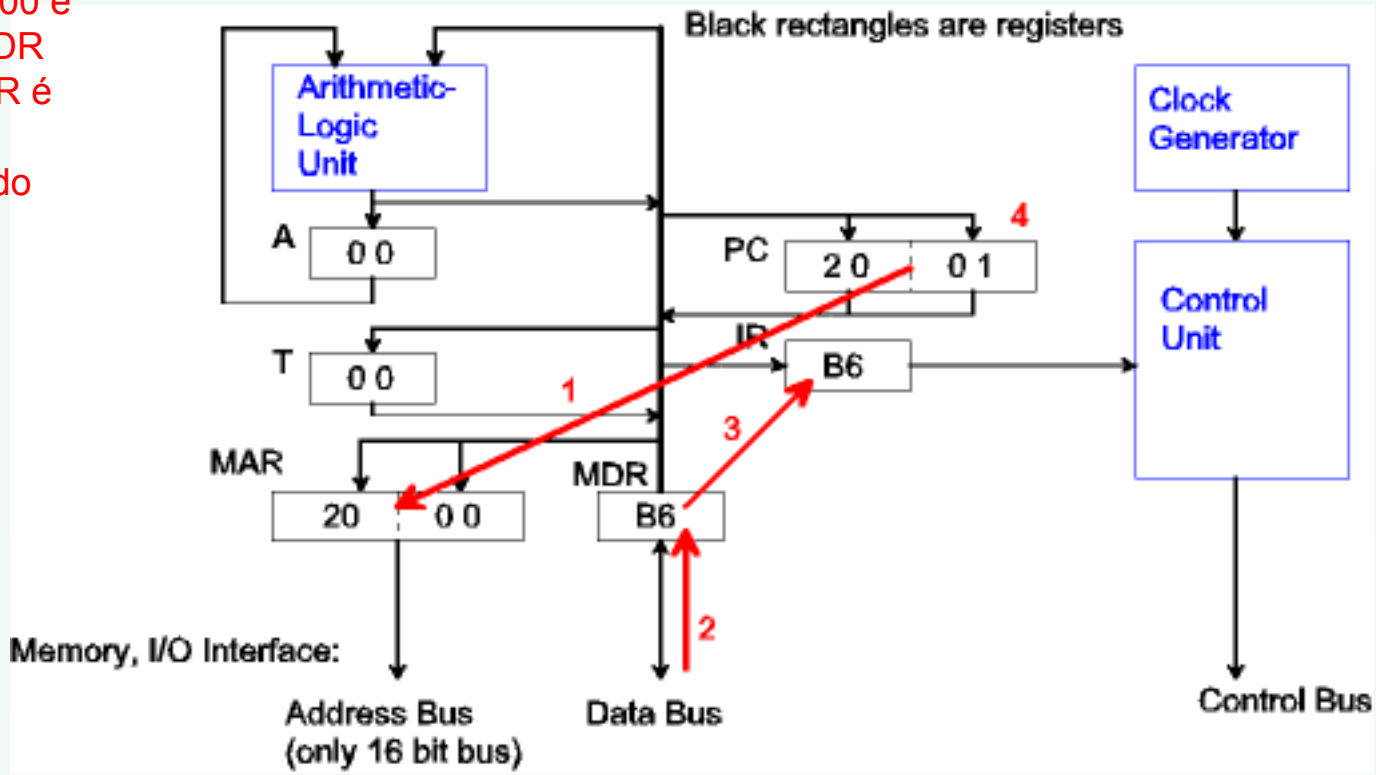
Execução de ‘LDAA’

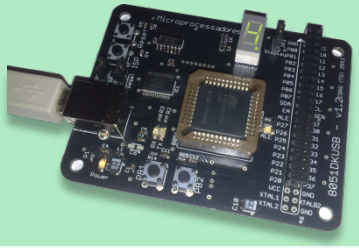




Execução de ‘LDAA’

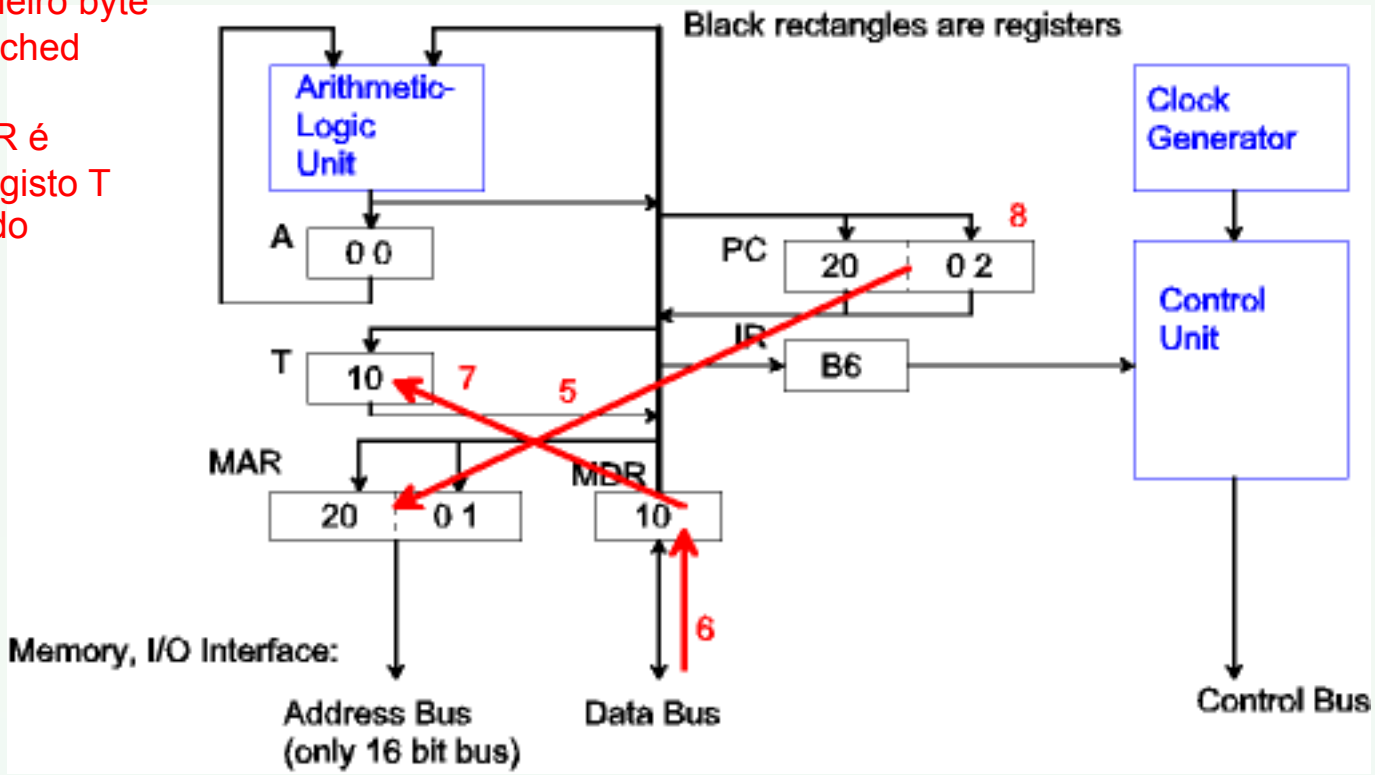
(1) PC é copiado para o MAR
(2) Conteúdo de $2000 é latched para o MDR
(3) Conteúdo de MDR é copied para IR
(4) PC é incrementado

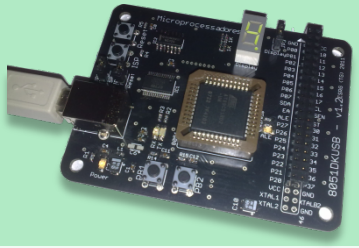




Execução de ‘LDAA’

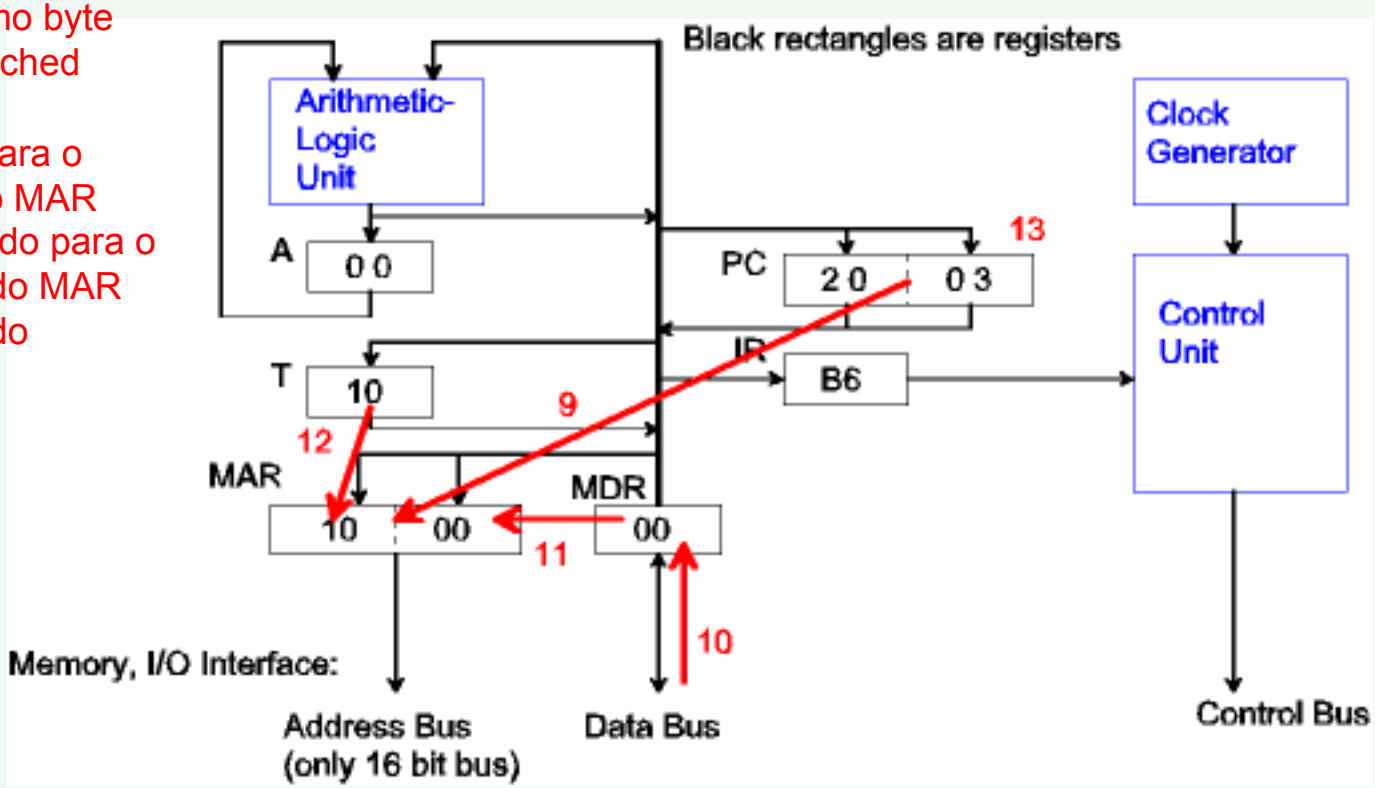
(5) PC é copiado para o MAR
(6) Conteúdo do primeiro byte do operando é latched para o MDR
(7) Conteúdo do MDR é copiado para o registo T
(8) PC é incrementado

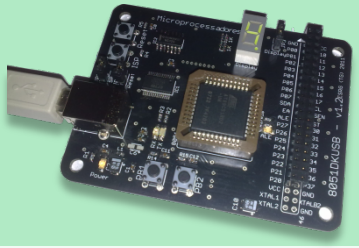




Execução de ‘LDAA’

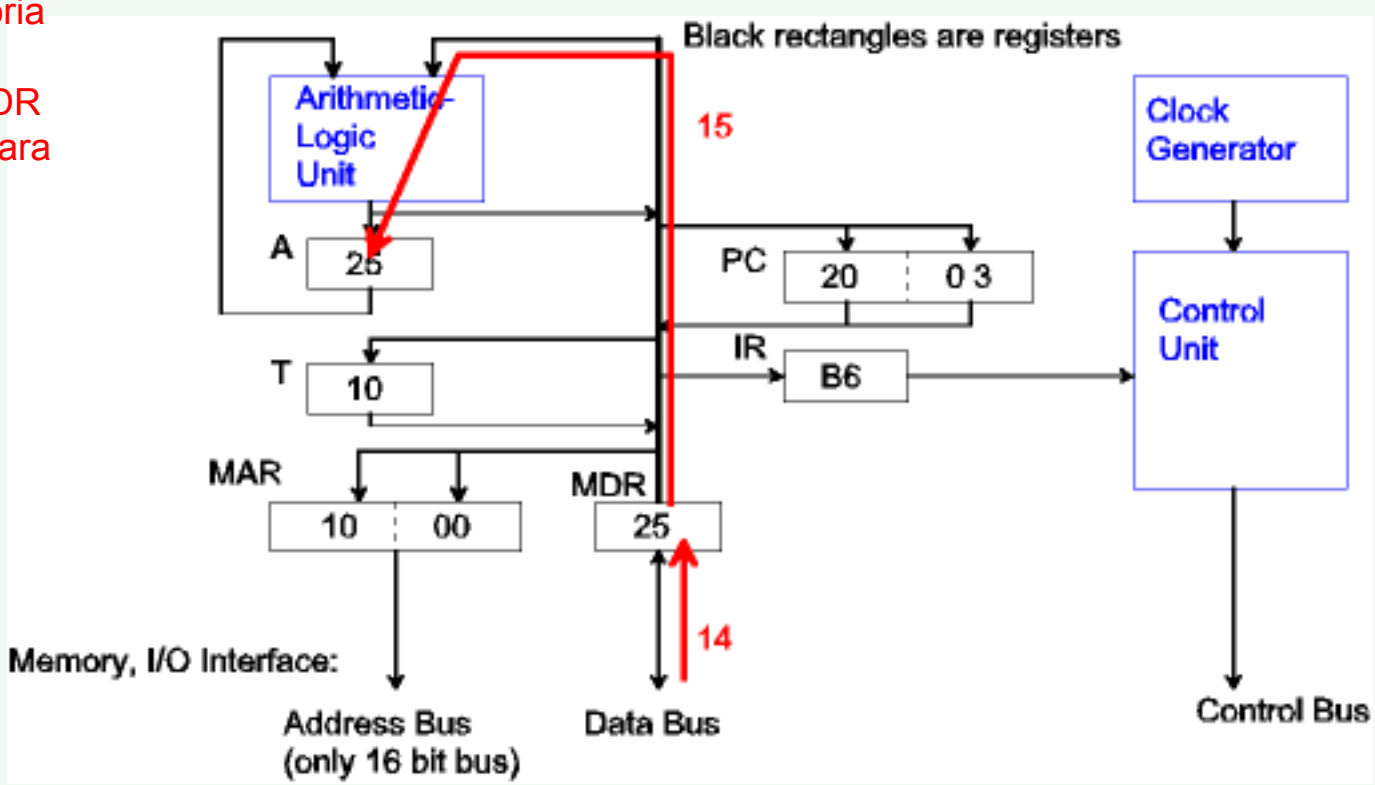
(9) PC é copiado para o MAR
(10) Conteúdo do último byte do operando é latched para o MDR
(11) MDR é copiado para o LSB (low byte) do MAR
(12) Registo T é copiado para o MSB (high byte) do MAR
(13) PC é incrementado

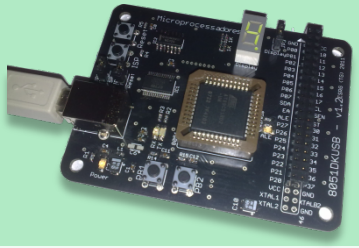




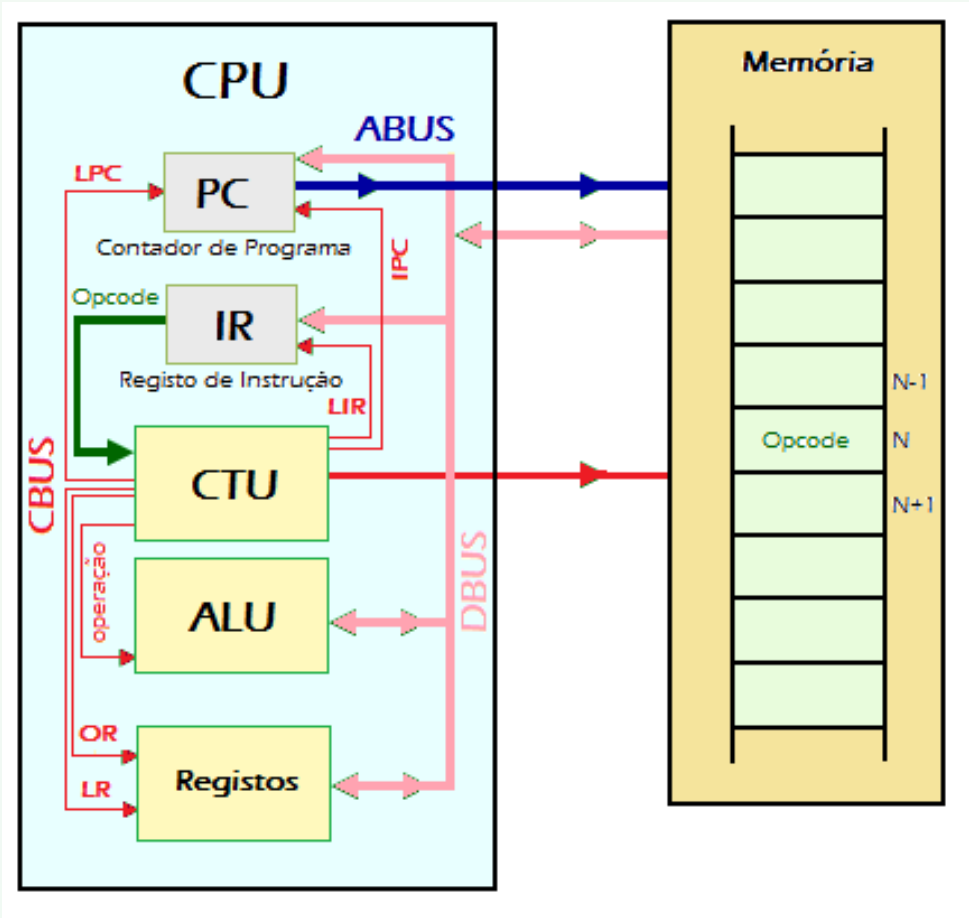
Execução de ‘LDAA’

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



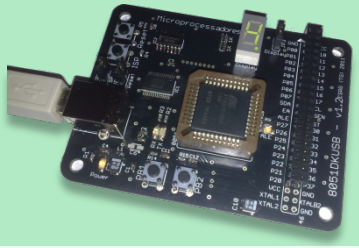


Fetch de um opcode



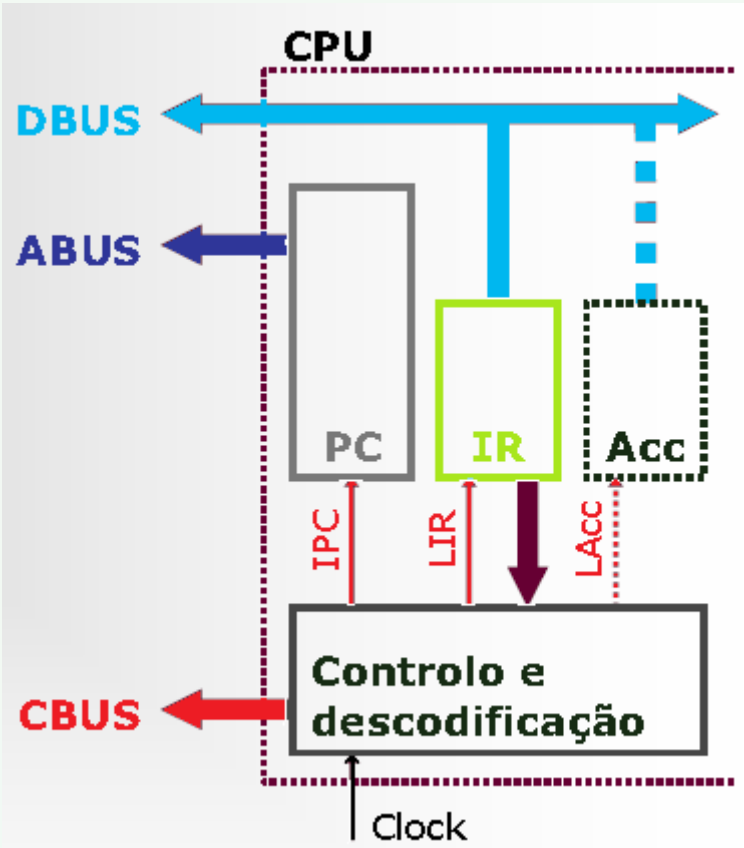
Passos

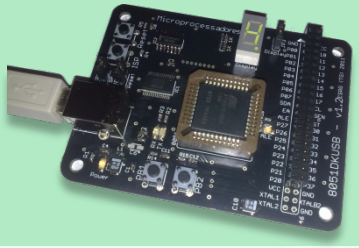
1. O conteúdo do PC é colocado no barramento de endereços;
2. O PC é incrementado para o *fetch* da próxima instrução;
3. A linha de controlo RD é activada;
4. A memória lê o conteúdo do endereço fornecido (código da instrução) e coloca-o no barramento de dados;
5. O *opcode* da instrução é transferido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





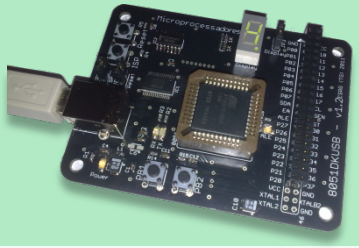
Executar uma instrução

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

Instrução no 8051	***Opcode* da Instrução**
MOV A,#160	74

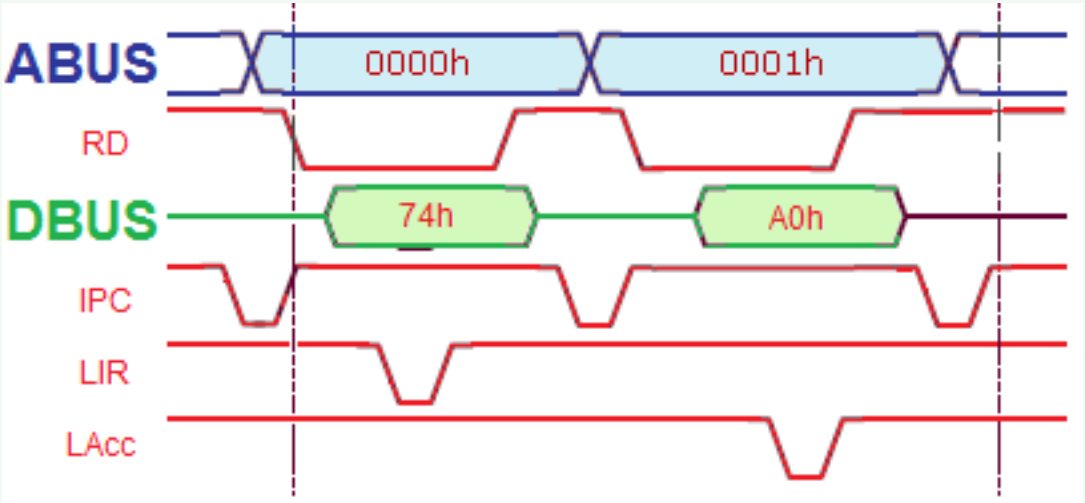
- A instrução será armazenada na memória de código. Há directivas *assembly* (instruções para o assemblador) que permitem definir em que endereço pretendemos armazenar a informação;

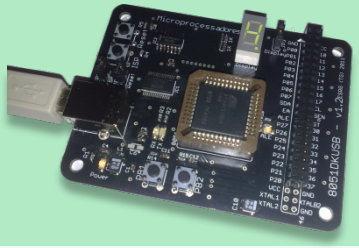
- Na instrução MOV A,#160 são armazenados dois bytes na memória de programa, o 74h que identifica a instrução “MOV A” e o A0h que é o valor constante que queremos carregar para o Acumulador. Estes dois bytes são armazenados sequencialmente na memória.



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).
- **Descodificar 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).





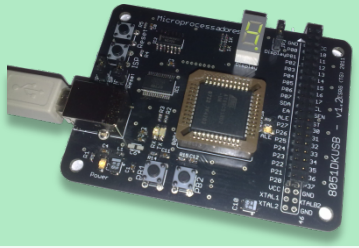
Classificação de processadores

(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*)

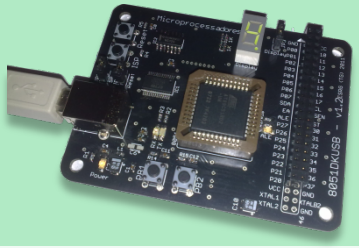
- **Contém informação necessária para interagir com o microprocessador, mas não os detalhes de como o microprocessador é projectado e implementado**

- **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*)

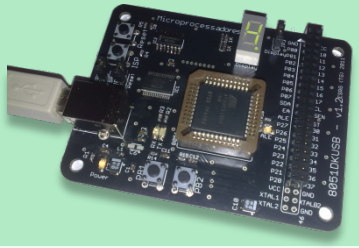
- **Fornece informações necessárias para a interacção com a memória**

  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**

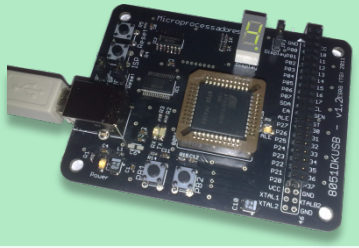
- ISA fornece instruções para activação/desactivação das interrupções

- Contudo, as acções físicas necessárias para reconhecer a ocorrência de uma interrupção externa não fazem parte do ISA



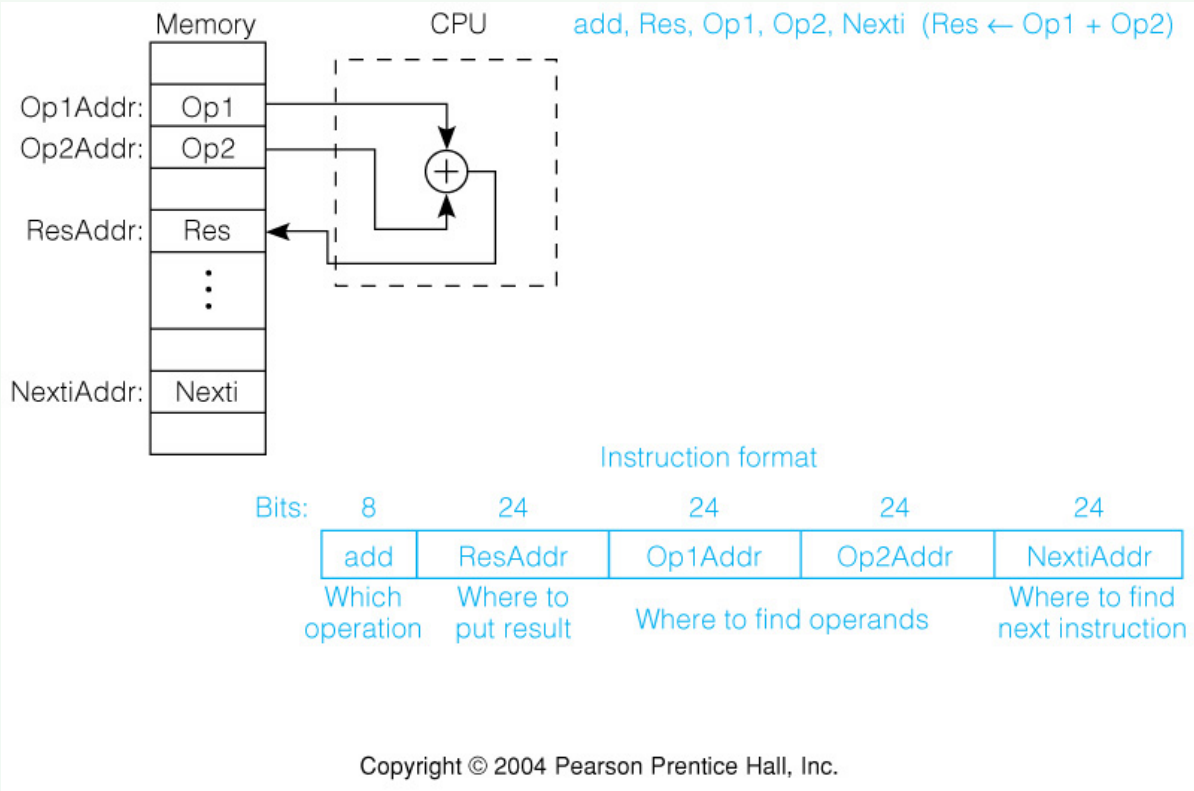
Opcode – Código de operação

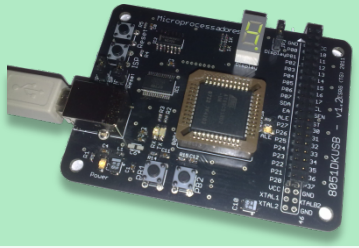
- **Que tipo de informação deve uma instrução especificar?**
  - **Operação a ser executada**
  - **Localização dos operandos**
  - **Localização de armazenamento do resultado**
  - **Localização da próxima instrução**



Máquinas de 4-endereços

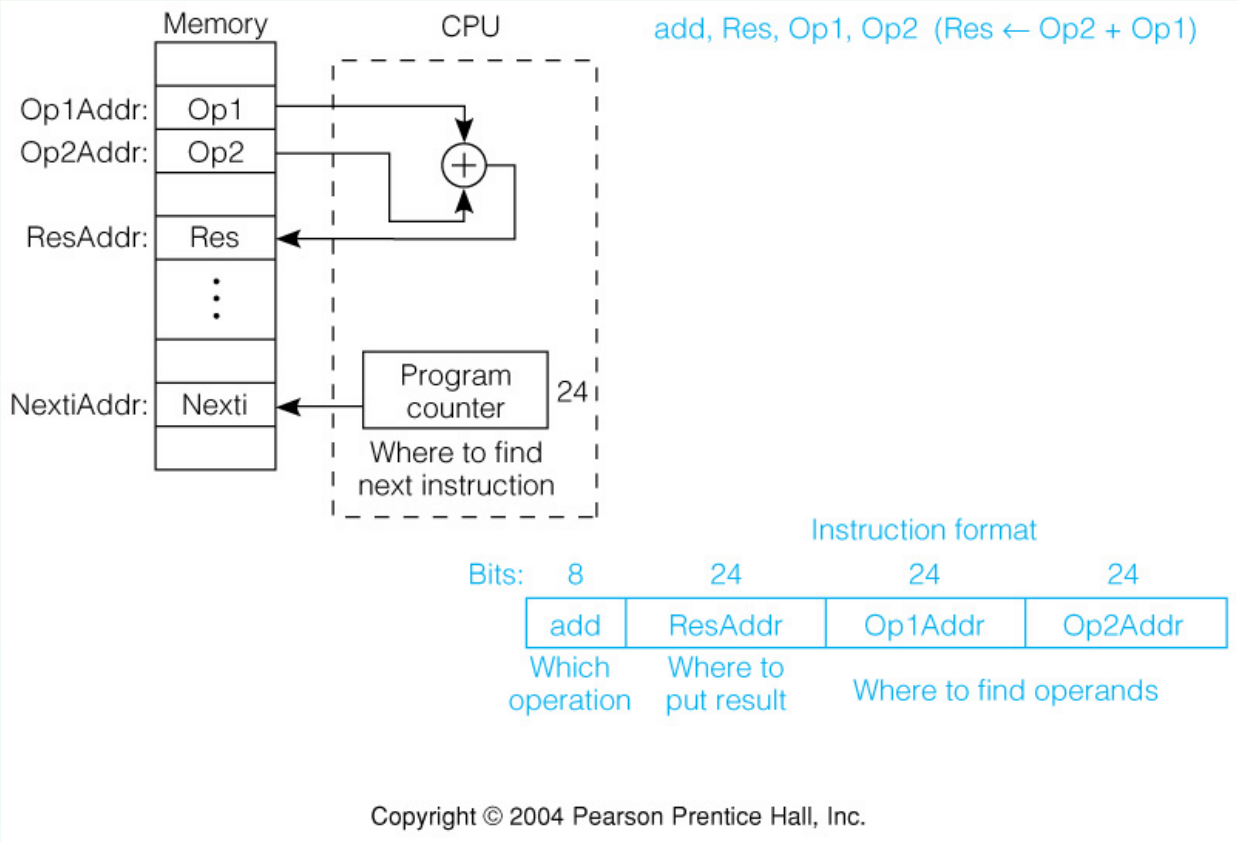
A instrução especifica de forma explícita os endereços dos operandos, do resultado e da próxima instrução

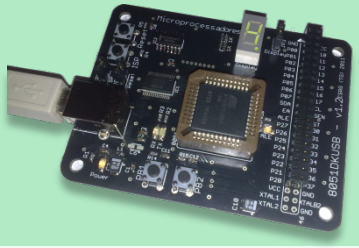




Máquinas de 3-endereços

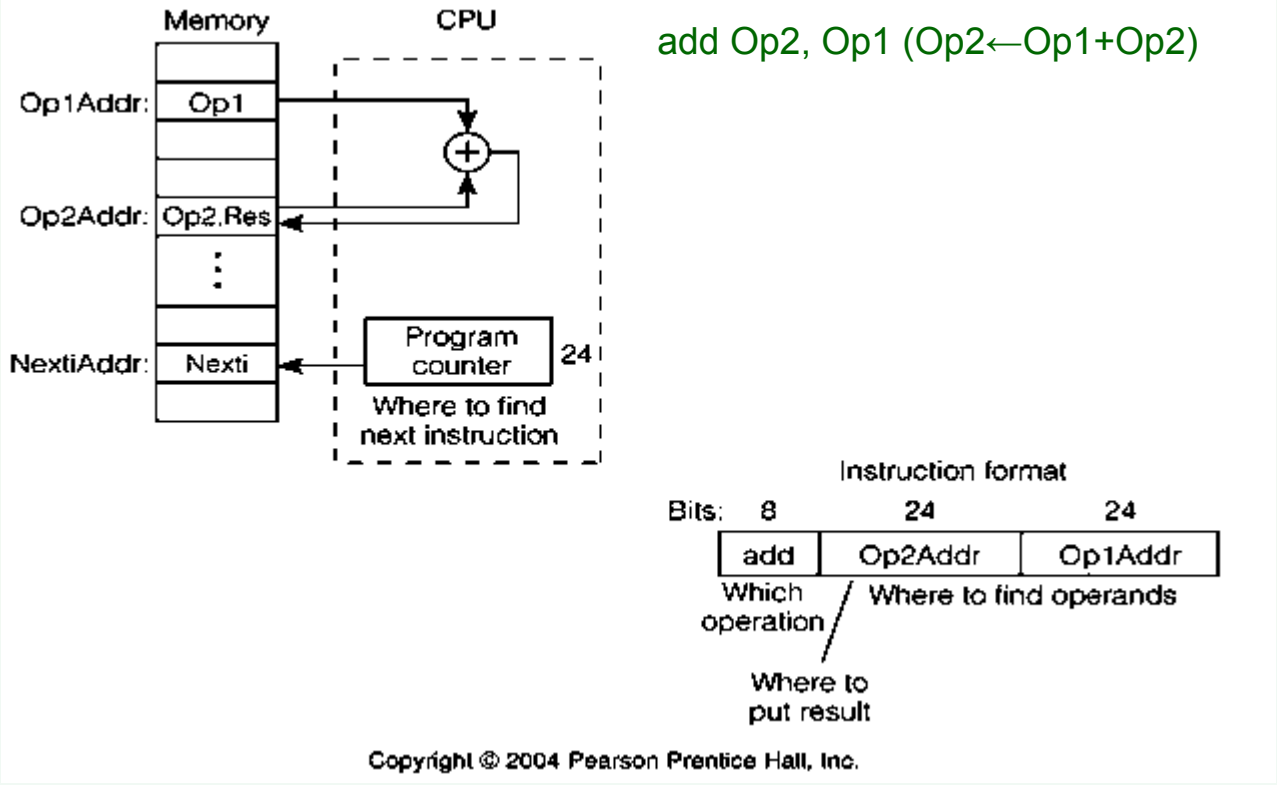
Endereço da próxima instrução (exceptuando as instruções de salto) é mantida num dos registos de estado do CPU – O *Program Counter* (PC)

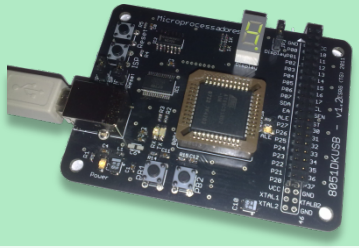




Máquinas de 2-endereços

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

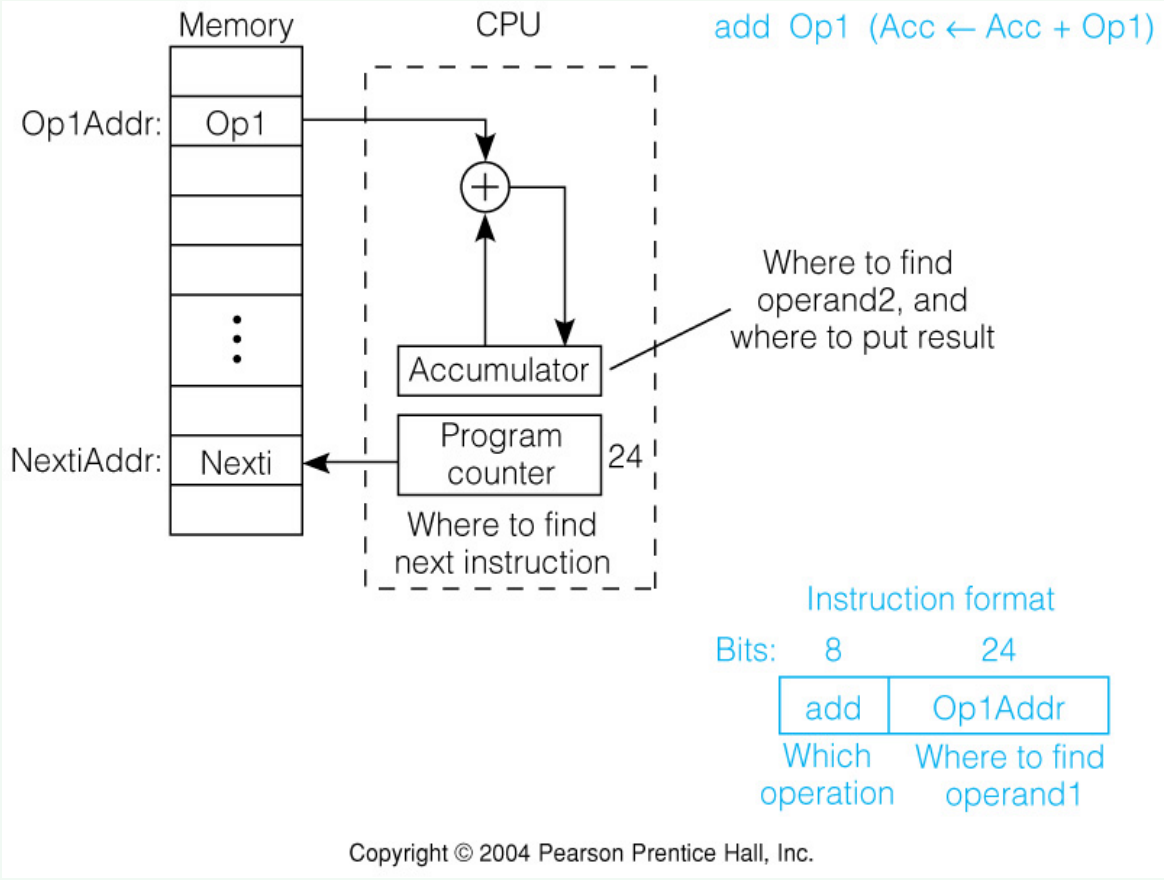


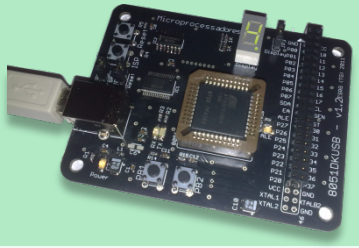


Máquinas de 1-endereço

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**

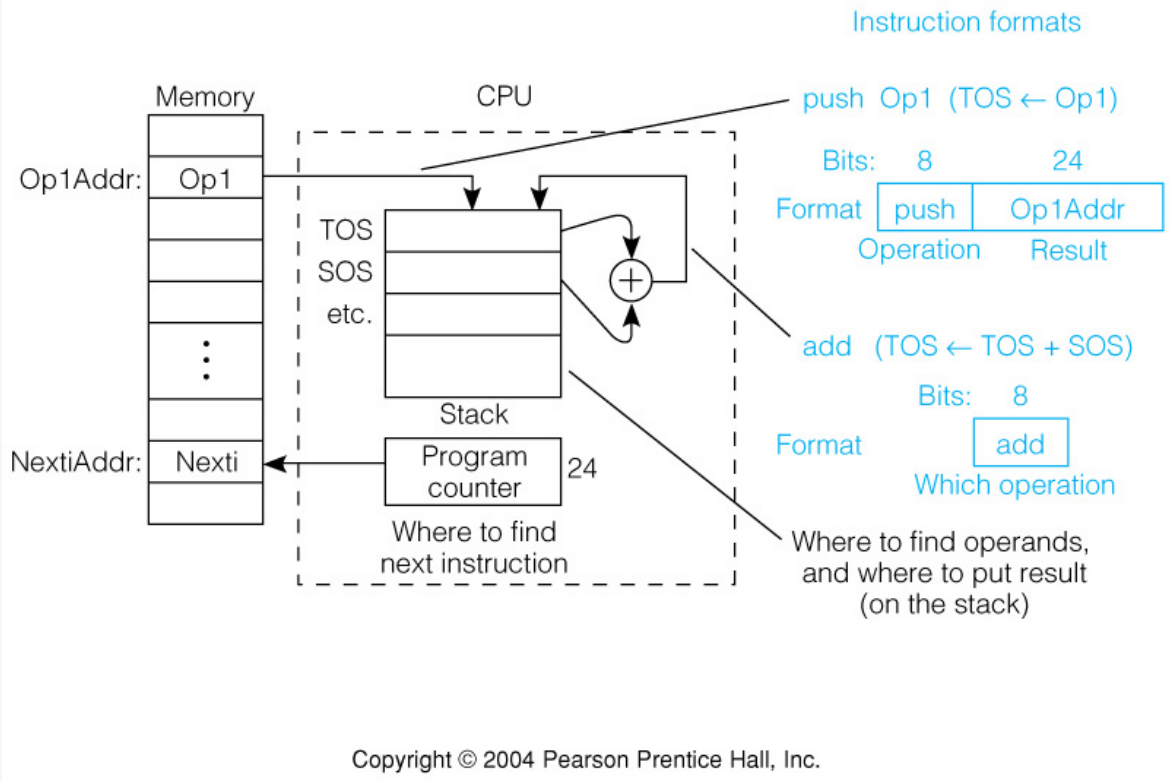




Máquinas de 0-endereço

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*)

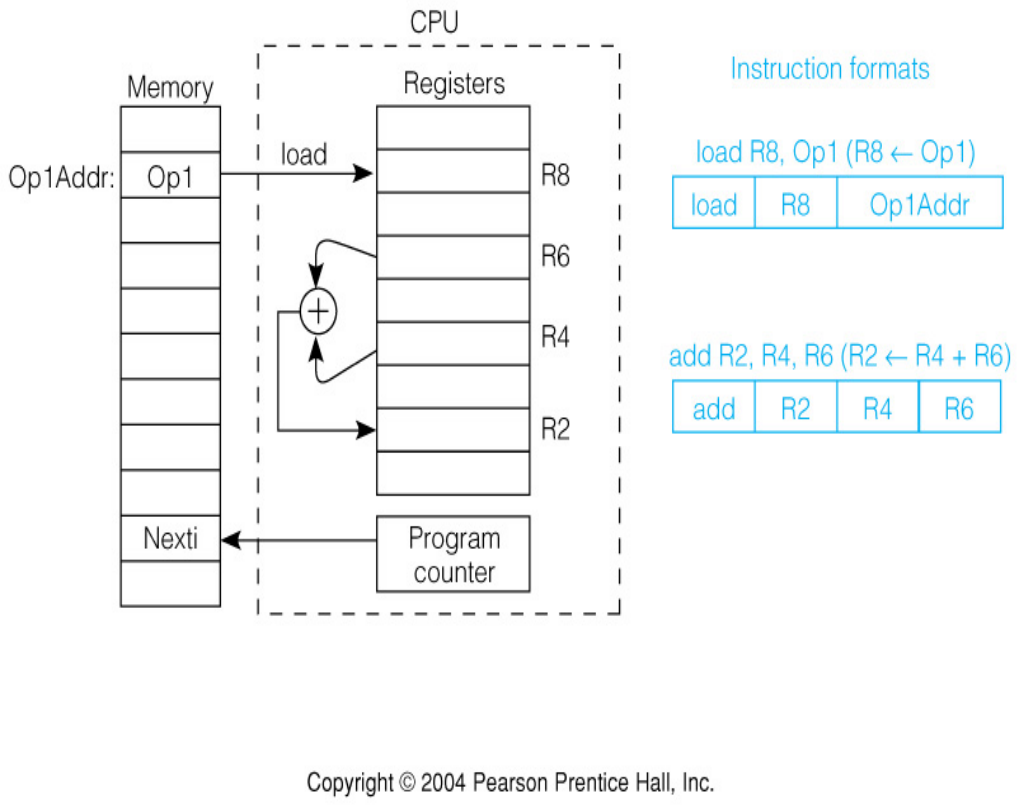


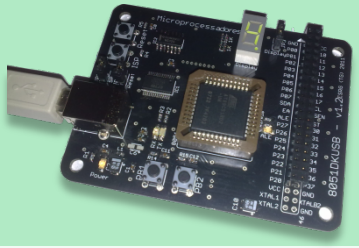
Máquinas de Registos genéricos (1-1/2-endereços)

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: 1-½ endereço**

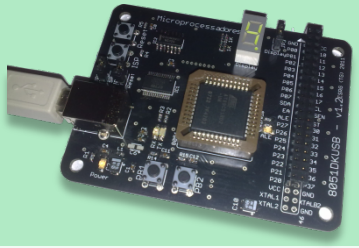
3. **Operações aritméticas de dois operandos especificam 3 meios-endereços**



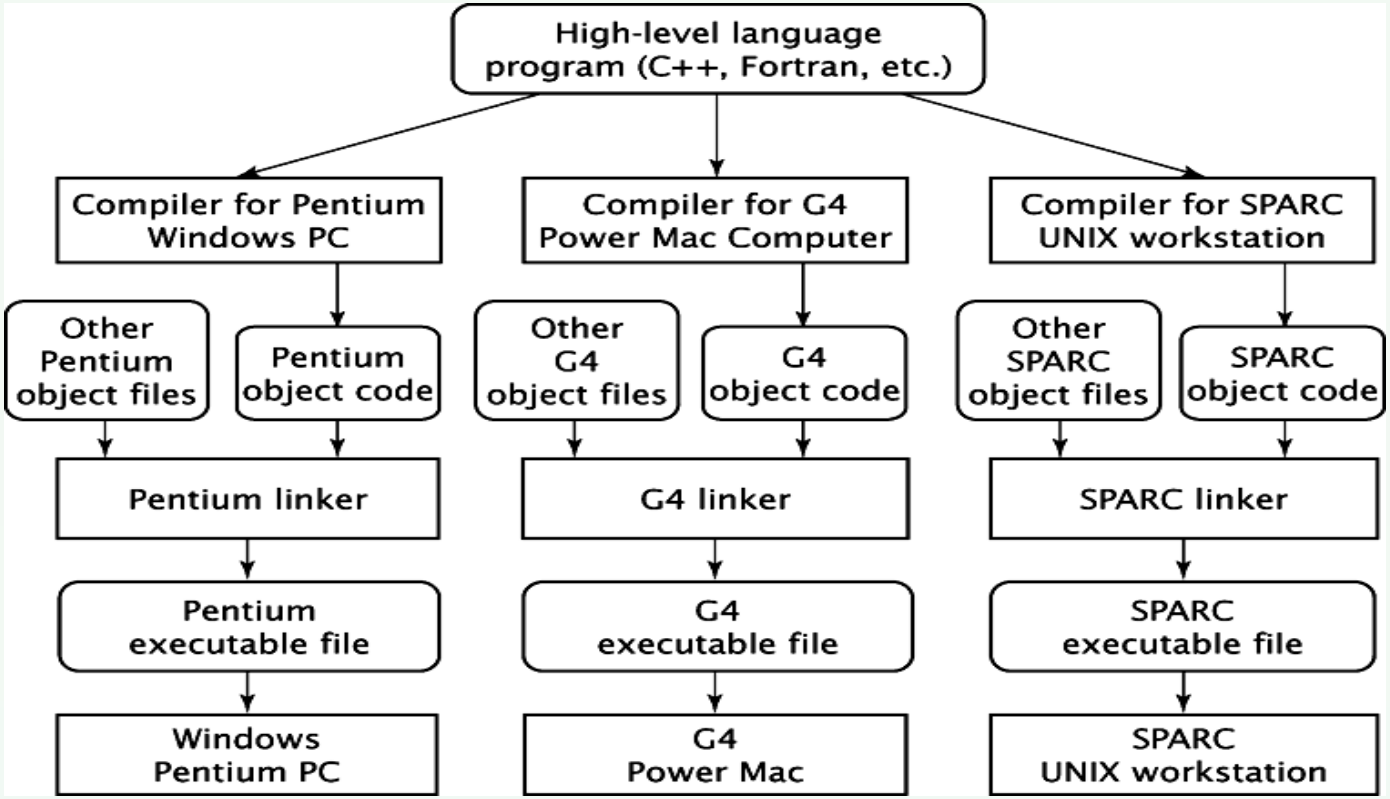


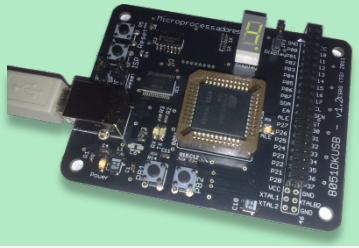
ISA e Níveis de linguagens de programação

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

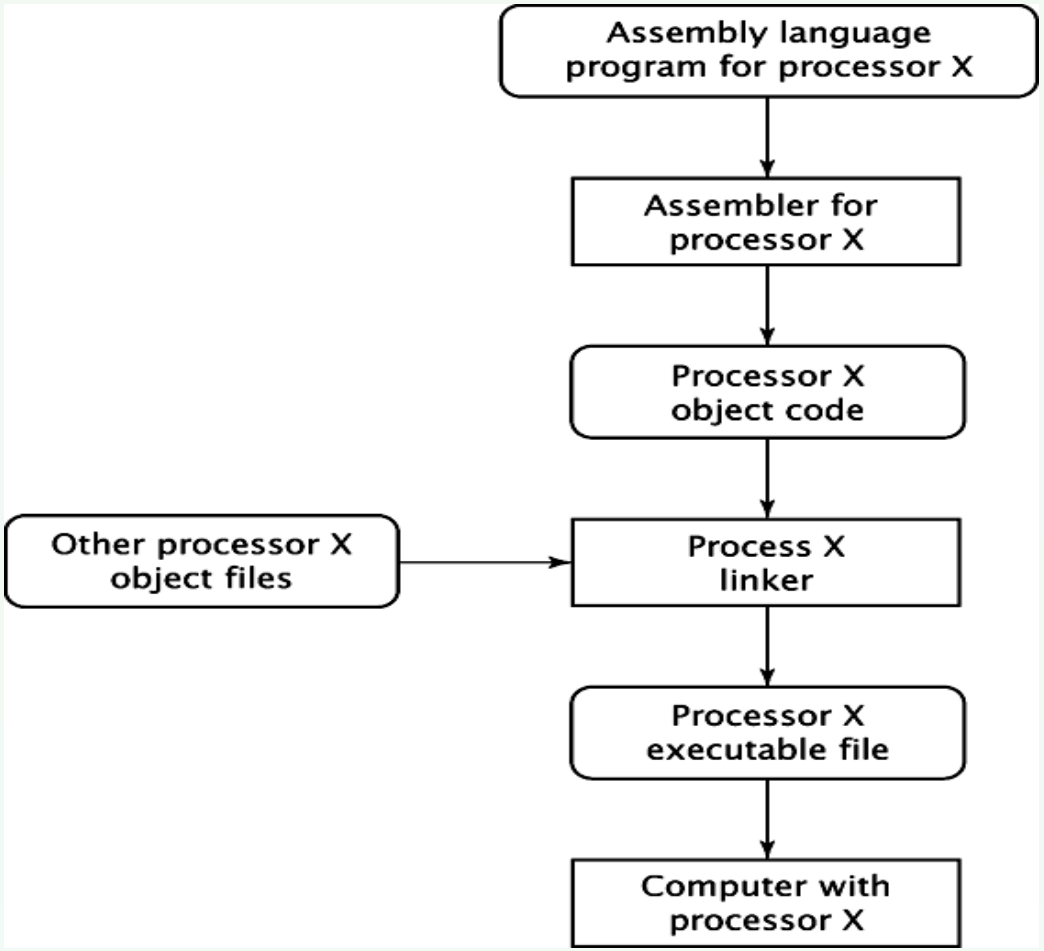


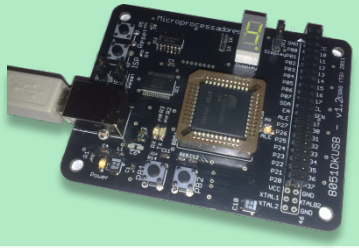
Processo de compilação





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

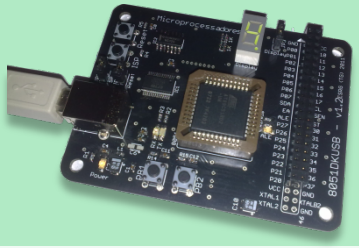
ii. Uma instrução *assembly* pode usar um dos vários modos de endereçamento para gerar este endereço

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

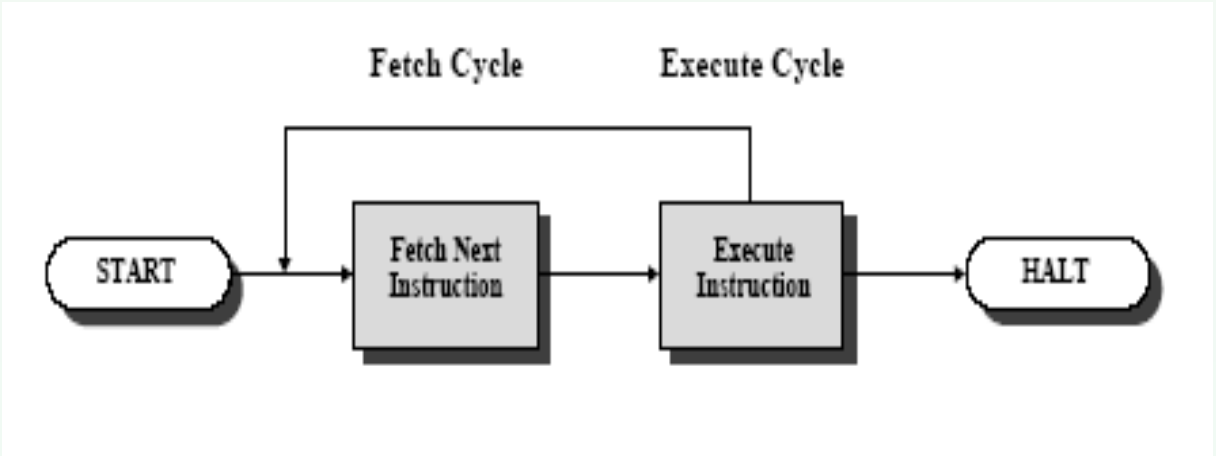
ii. Um μP pode apresentar um único formato para todas as instruções, ou então vários formatos diferentes, tendo cada instrução apenas um formato para o código da instrução

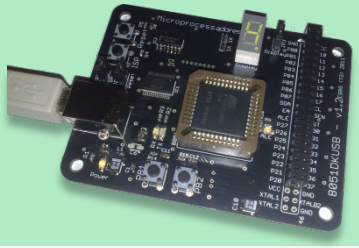


Ciclo de instrução

- 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



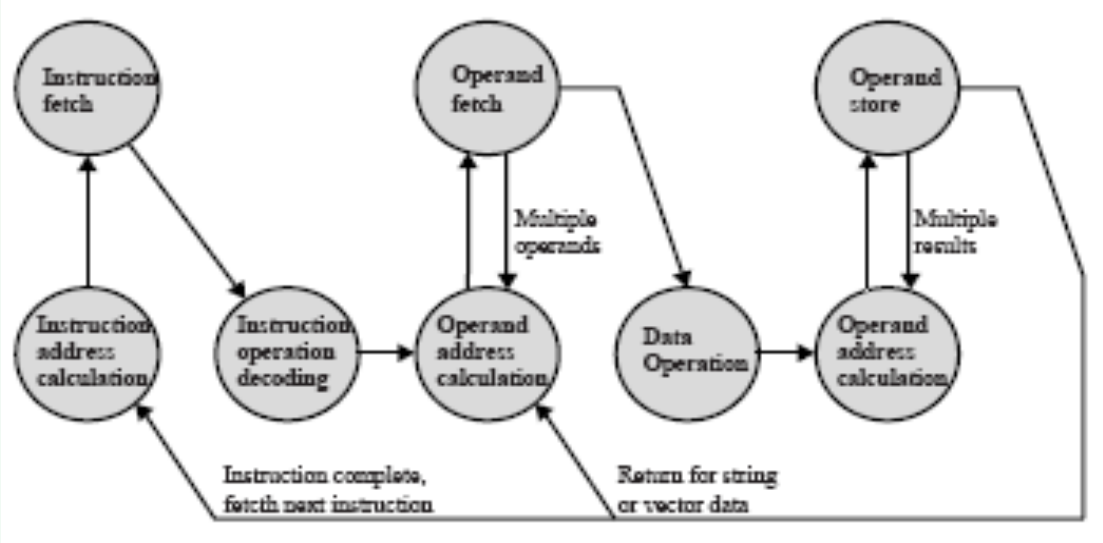


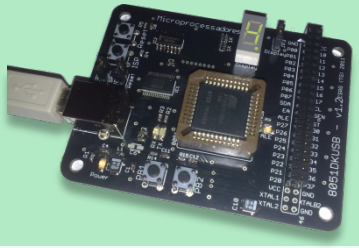
Ciclo de instrução

- Durante a descodificação 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

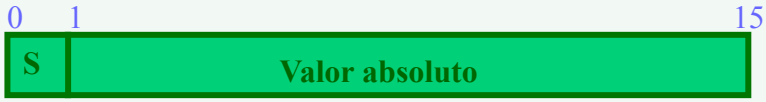




Computador Imaginário

- Formato da Instrução

- Formato para número inteiros

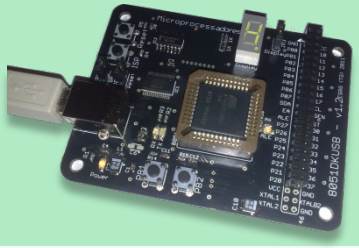


- Registos internos do CPU
  - PC (*Program Counter*) = Endereço da instrução
  - IR (Instruction Register) = Instrução em execução
  - AC (Accumulator) = Armazenamento temporário
- Lista parcial de *opcodes*
  - 0001 = carregar para AC o conteúdo de uma célula de memória
  - 0010 = Armazenar conteúdo de AC na memória
  - 0101 = adicionar conteúdo de AC com o conteúdo de uma célula de memória

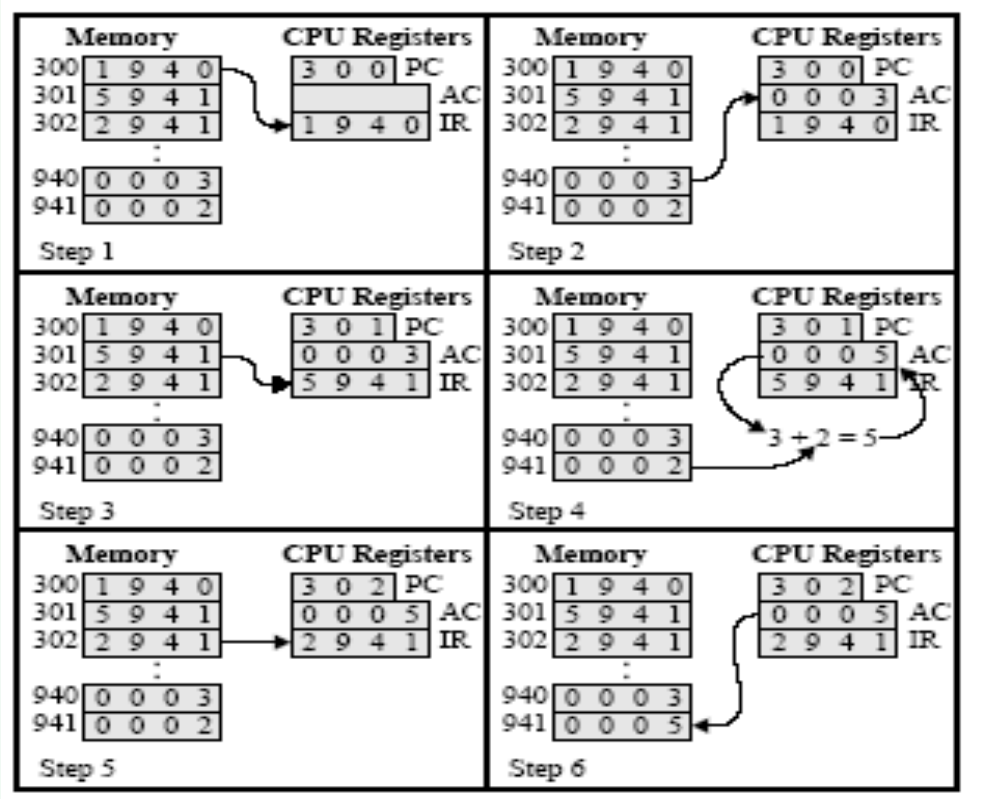
Como organizava a memória?

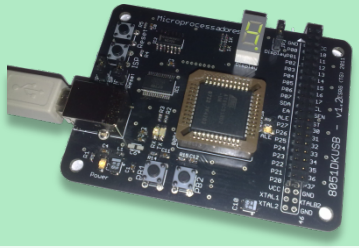
Quantas células de memória podem ser directamente endereçáveis?

Quantos *opcodes* diferentes pode ter o ISA?



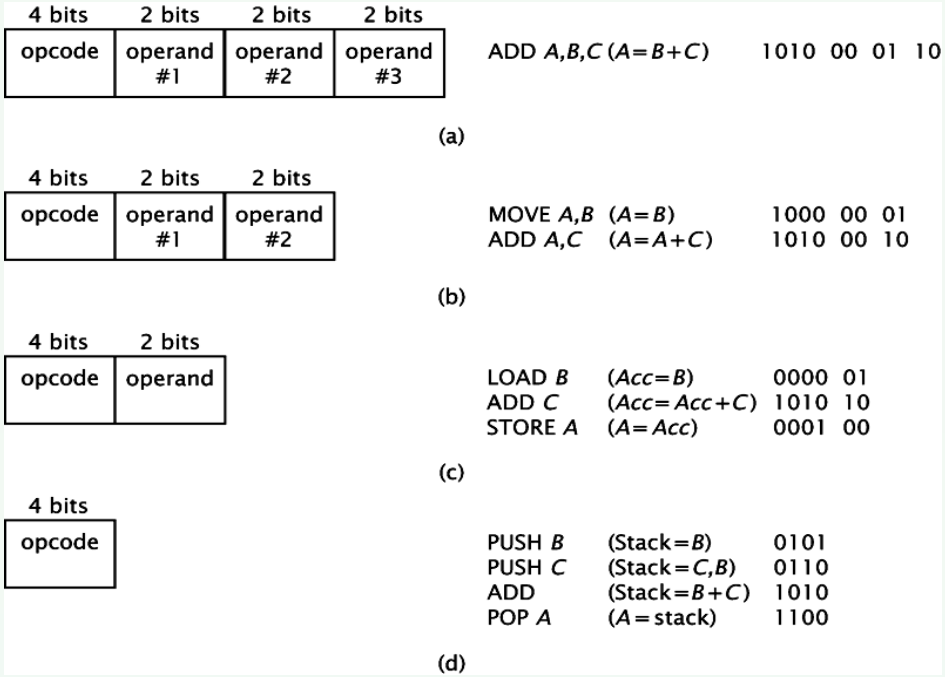
- **Execução parcial de um programa sobre o computador imaginário**





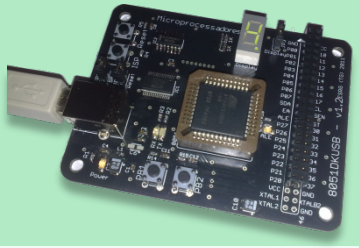
Formato da instrução

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



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



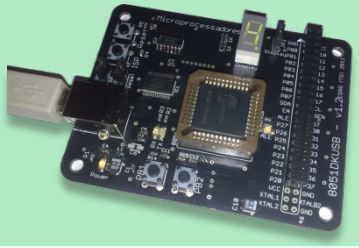
Projecto ISA

- **Aspectos a considerar ao projectar ISA**

1. Que funcionalidades se pretende da ISA e do processador?
2. Plenitude

   Será que o conjunto de instruções contém todas as instruções necessárias para programar todas as tarefa pretendidas?
3. Ortogonalidade
   1. Instruções são ortogonais caso não sejam redundantes
   2. Um conjunto de instruções bem projectado deve minimizar as redundâncias entre instruções, fornecendo deste modo ao programador apenas as funcionalidades pretendidas sob a forma de um número reduzido de instruções
4. Qual o número ideal de registos internos que permitam optimizar a ISA?

   O μP deve fornecer registos suficientes de modo a minimizar acessos à memória, melhorando deste modo o desempenho



Projecto ISA

6. Quais os tipos e dimensões dos dados que o μP processará?

7. Será necessário garantir *backward compatibility* com outros μP?

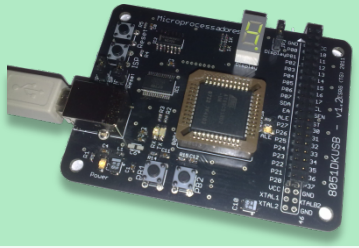
8. As interrupções são necessárias?

   Mecanismos de *polling* podem não ser tão eficientes?

9. As instruções de salto condicionais são necessários?

   A ISA poderá ter que fornecer as *flags* (registos de 1-bit) para a memorização das várias condições

Como exemplo analisar o paragráfo 3.4 com a descrição do "*Relatively Simple Instruction Set Architecture*"



Projecto ISA: Resumo

Ao projectar um ISA deve-se descrever:

1. O estado do processador e da memória

   Dimensões e números dos registos internos, organização da memória, estado de entrada/ saída se aplicável

2. Formato e interpretação dos dados nos registos: significado dos campos de registos

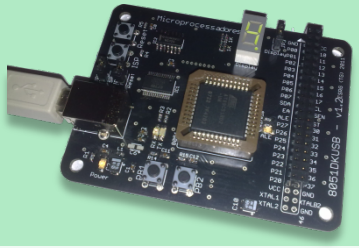
   Tipos de dados, formatos da instrução, interpretação do endereço efectivo

3. Interpretação da instrução: acções efectuadas por todas as instruções

   Especificação do ciclo *fetch-execute* e gestão das interrupções

4. Execução da instrução: comportamento individual das instruções

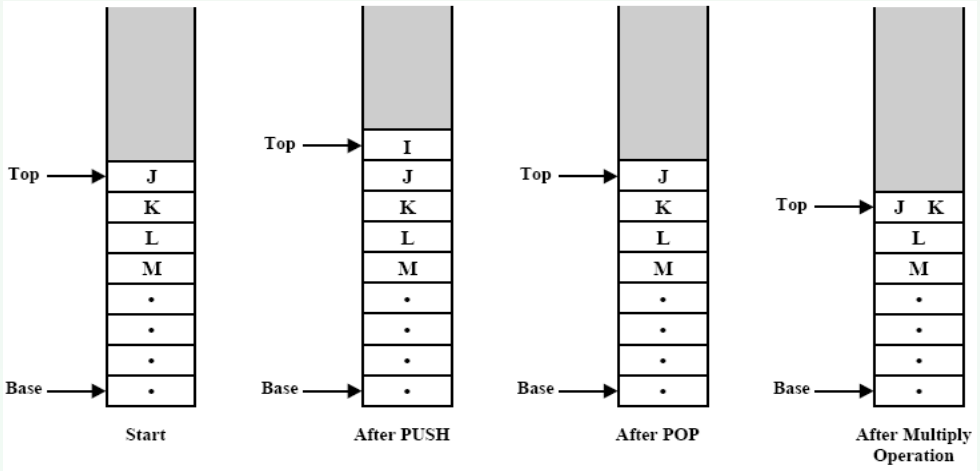
   Especificando as classes de instruções em movimento de dados, salto, operações aritméticas e outros (para as instruções que não pertencem a nenhumas das classes anteriores), assim como acções efectuadas por cada instrução

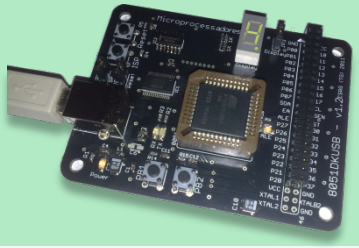


Stack - Pilha

- **Stack (Pilha)**

1. A pilha é um conjunto ordenado de elementos
2. Num dado instante, apenas um elemento pode ser acedido
3. O ponto de acesso é designado por topo da pilha
4. É uma estrutura do tipo LIFO (*Last-In-First-Out*), em que os itens apenas podem ser inseridos (PUSH) ou removidos (POP) a partir do topo da pilha
5. O comprimento da pilha é variável





Stack - Pilha

- **Operações efectuadas sobre a pilha**

1. PUSH

   Insere um novo elemento a partir do topo da pilha e actualiza o topo da pilha

2. POP

   Remove o elemento presente no topo da pilha e actualiza o topo da pilha

3. Operação de um único operando

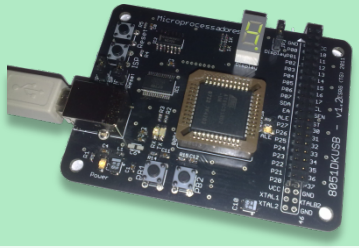
   Efectua a operação sobre o elemento presente no topo da pilha e substitui o elemento do topo da pilha com o resultado

4. Operação de dois operandos

   1. Efectua a operação sobre dois elementos do topo da pilha
   2. Remove os dois elementos do topo da pilha, actualizando o topo da pila
   3. Coloca o resultado no topo da pilha e actualiza o topo da pilha

5. Qual o modo de endereçamento usado?

   Todas as operações referenciam uma única localização (o topo da pilha), pelo que o endereço do(s) operando(s) está/estão implícito na instrução



Stack - Pilha

- **Implementações da pilha**
  1. Pilha acessível ao programador

     A ISA deve fornecer as operações como parte do conjunto de instruções

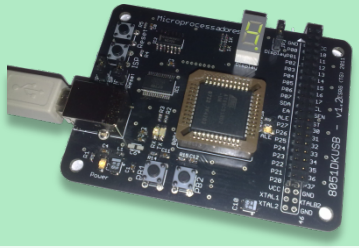
  2. Pilha apenas usado pelo CPU

     A ISA não fornece as operações como parte do conjunto de instruções, visto que estas operações apenas serão usadas na gestão das chamadas/ retorno dos procedimentos/funções

  3. Um conjunto de localizações contíguas serão reservadas na memória principal ou memória virtual para armazenar os elementos da pilha

  4. Três endereços, normalmente, armazenados nos registos do CPU são necessários para garantir o funcionamento adequado das operações

     1. Apontador para o topo da pilha (*Stack pointer*)
     2. Endereço base da pilha (*Stack base*)
     3. Limite da pilha (*Stack limit*)



Stack - Pilha

- **Implementações da pilha**
  1. Normalmente, a pilha cresce a partir dos endereços superiores para endereços inferiores

     Isto é, o limite é dado pelo endereço inferior e o topo da pilha é dado pelo endereço superior
  2. Normalmente, para acelerar operações sobre a pilha, costuma-se manter os dois elemento do topo da pilha em registos e apontar o topo da pilha para o terceiro elemento