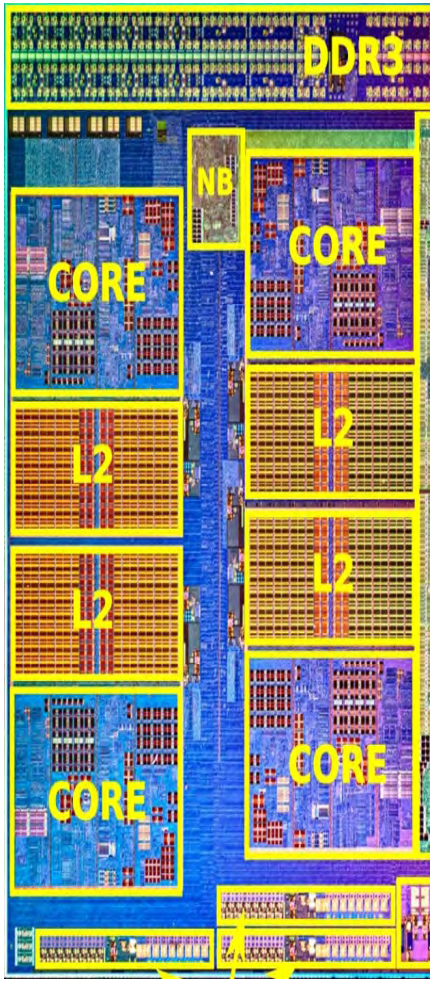
Arquitetura e Organização de Computadores 1



Fonte: http://www.techspot.com/article/904-history-of-the-personal-computer-part-5/

ISA: Arquitetura do Conjunto de Instruções
Parte 1

Luciano de Oliveira Neris
luciano@dc.ufscar.br

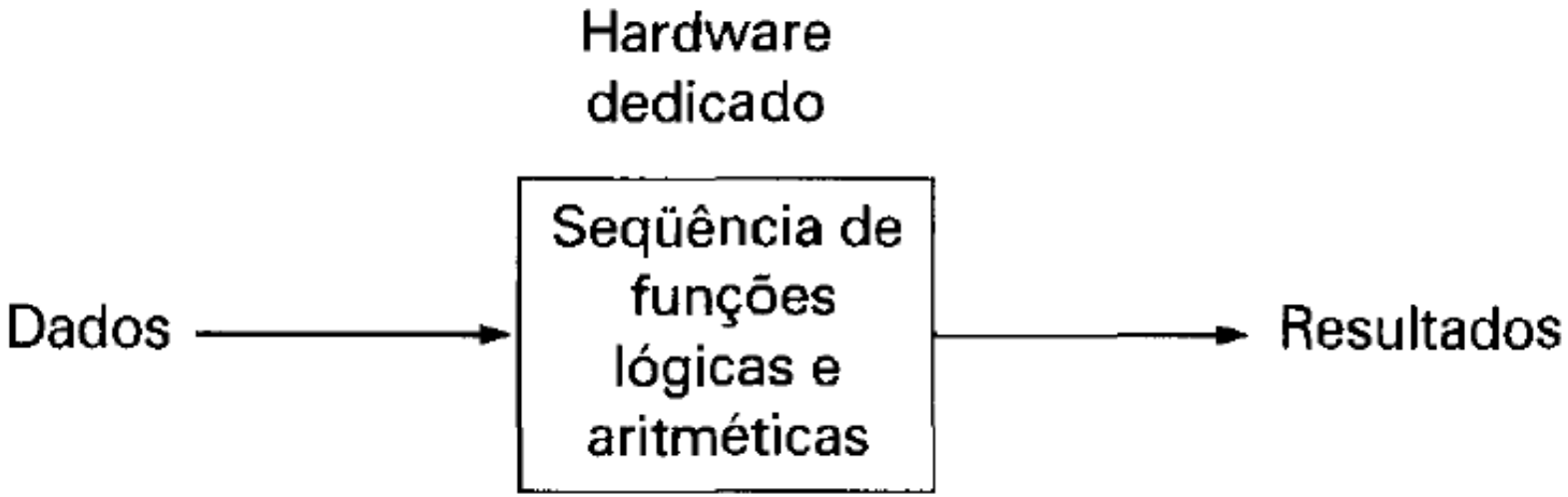
Adaptado de slides do prof. Marcio Merino Fernandes

Departamento de Computação
Universidade Federal de São Carlos

Computador

- Composto de um pequeno conjunto de componente lógicos básicos
  - Podem ser combinados para armazenar dados binários e executar operações aritméticas e lógicas sobre esses dados.
- Para cada aplicação particular pode-se obter uma configuração de componentes lógicos projetada especificamente para uma dada aplicação
  - Hardware é dedicado para uma aplicação particular, o sistema apenas lê dados e produz resultados

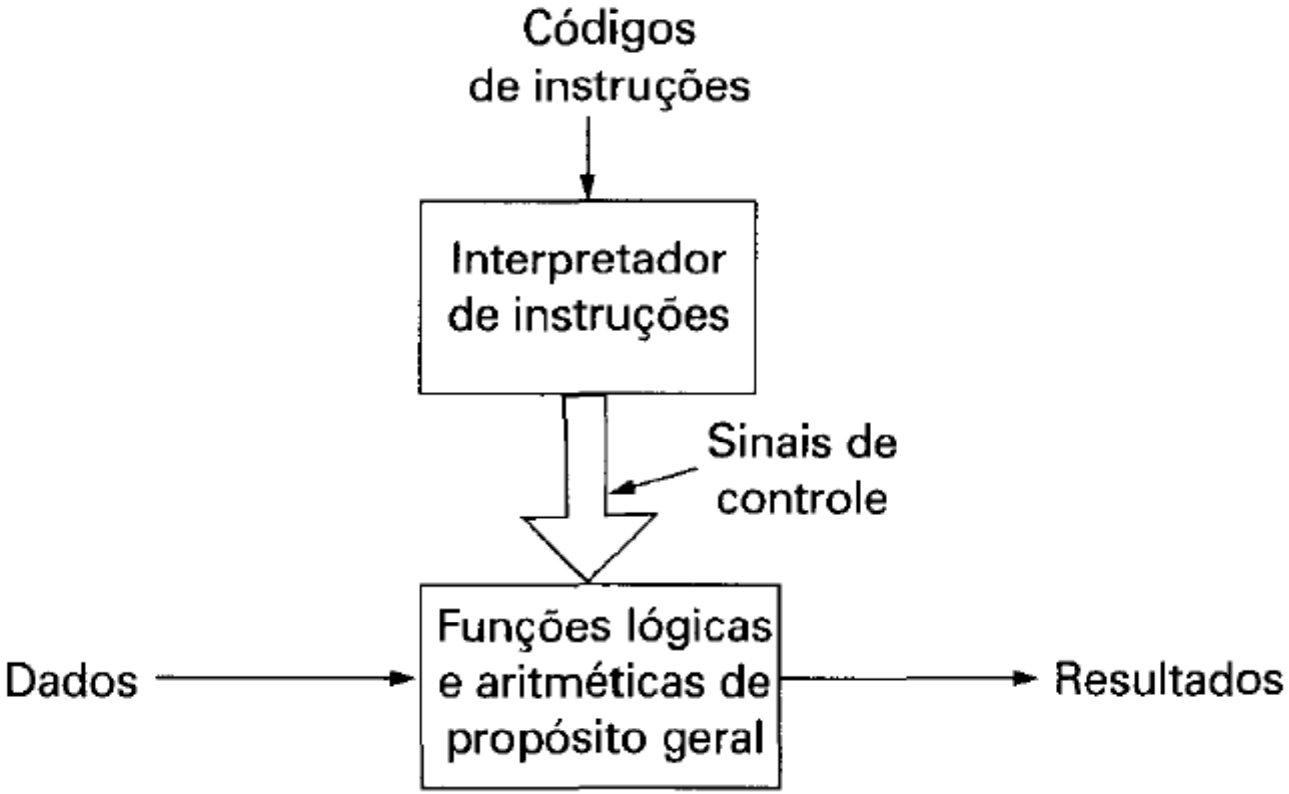
Computador


Hardware
dedicado
Seqüência de
funções
lógicas e
aritméticas
Dados
Resultados

Computador

- Alternativa: configuração de funções lógicas e aritméticas de propósito geral
  - Conjunto de componentes de hardware capaz de executar várias funções sobre os dados, dependendo dos sinais de controle que lhe são aplicados.
- Hardware de propósito geral é capaz de ler dados e sinais de controle e produzir resultados.
  - Não é necessário projetar um novo hardware para cada aplicação nova, o programador simplesmente precisa fornecer um novo conjunto de sinais de controle.

Computador


Códigos
de instruções
Interpretador
de instruções
Sinais de
controle
Dados
Funções lógicas
e aritméticas de
propósito geral
Resultados

Computador

- Componentes importantes: módulo de interpretação de instruções e o módulo de execução das funções lógicas e aritméticas

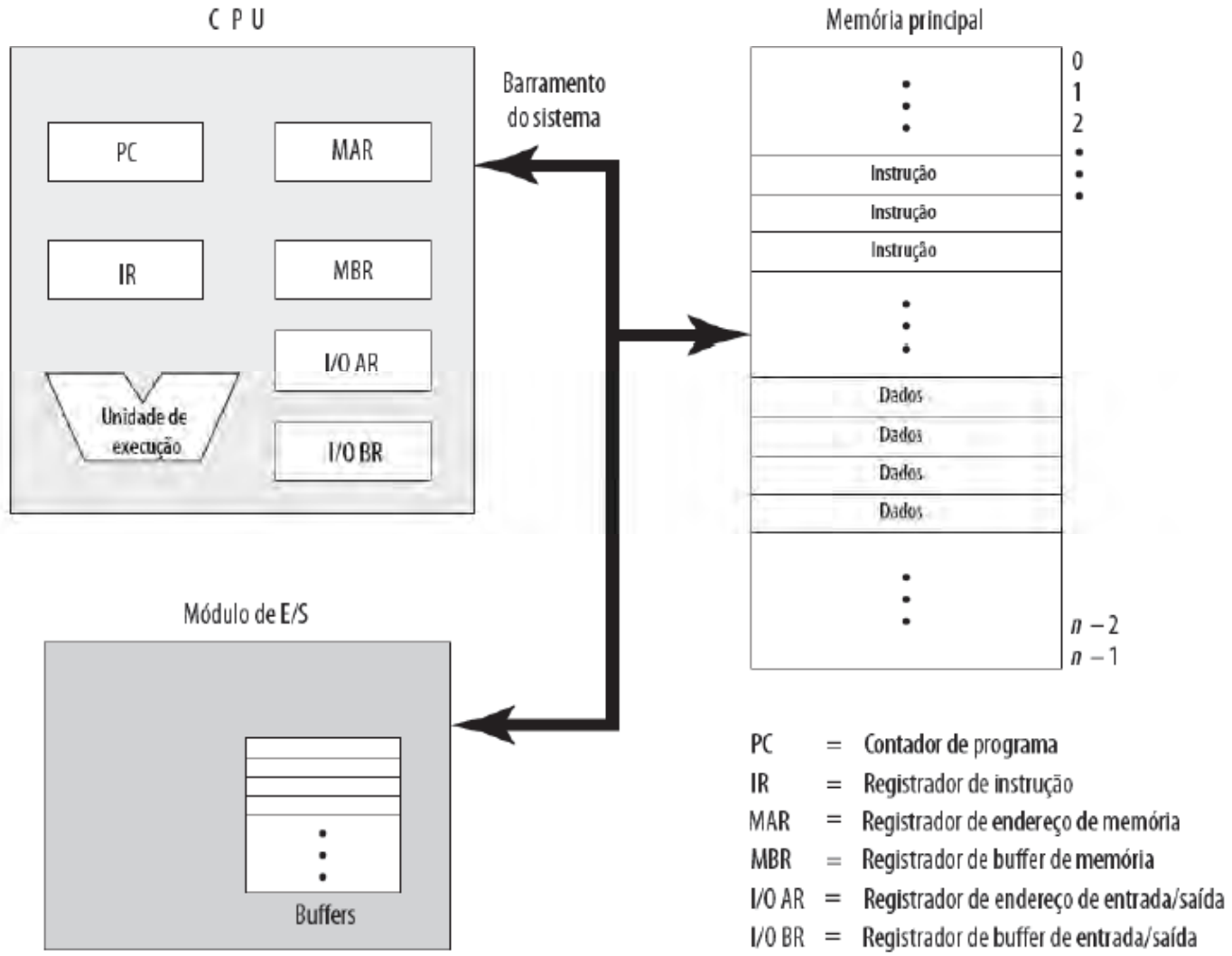
- Outros componentes: módulos de E/S (I/O), memória

  - Entrada: introduzem os dados e instruções no sistema, recebendo-os em algum formato e os converte em uma representação interna.

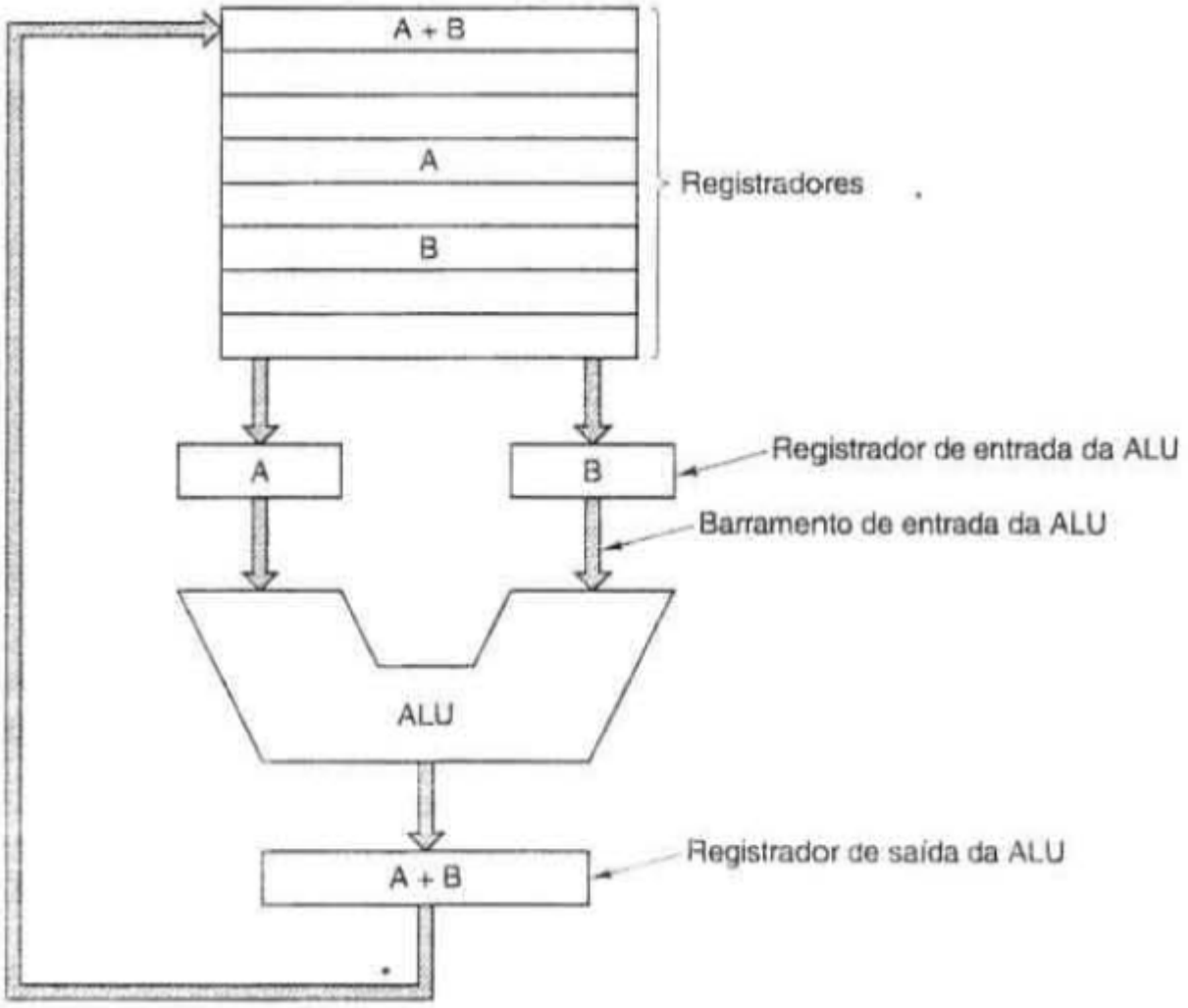
  - Saída: apresentar ou armazenar os resultados produzidos

  - Armazenamento: armazena instruções e dados temporariamente.

Computador


C P U
PC
MAR
IR
MBR
I/O AR
I/O BR
Unidade de execução
Barramento do sistema
Memória principal
0
1
2
Instrução
Instrução
Instrução
Dados
Dados
Dados
Dados
n – 2
n – 1
Módulo de E/S
Buffers
PC = Contador de programa
IR = Registrador de instrução
MAR = Registrador de endereço de memória
MBR = Registrador de buffer de memória
I/O AR = Registrador de endereço de entrada/saída
I/O BR = Registrador de buffer de entrada/saída

Computador - Datapath



Computador

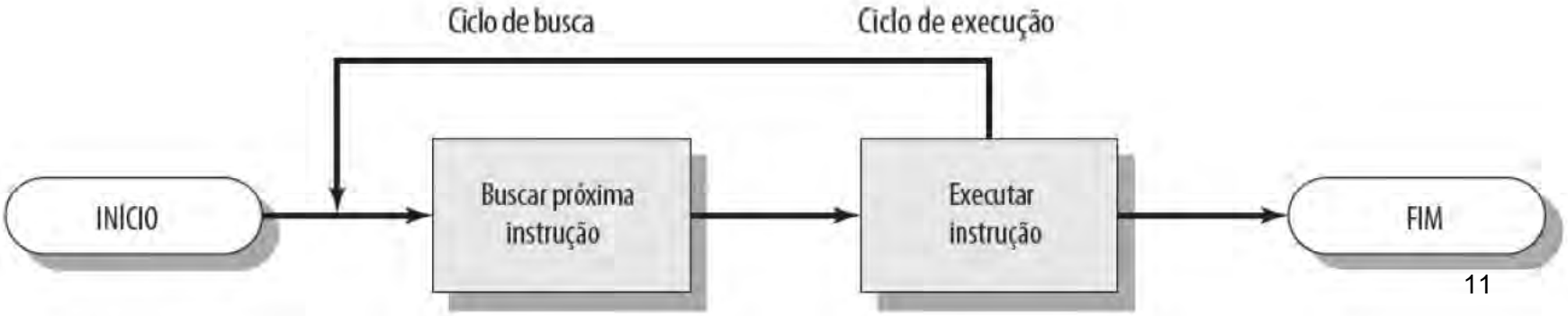
- Função básica: executar um programa que é constituído por um conjunto de instruções armazenadas na memória.
  - O processador realiza o trabalho efetivo de executar as instruções especificadas no programa.

- Passos para o processamento de instruções:
  - Busca (fetch): o processador lê instruções na memória (uma de cada vez),
  - Execução: executa cada instrução lida da memória

- A execução de um programa consiste na repetição desse processo de busca e execução de instruções.

Computador

- No contador de programas (PC) estará o endereço da próxima instrução a ser executada
- O processador faz a busca da instrução na posição de memória que está armazenada no PC
- O PC é incrementado
  - A não ser que a próxima instrução não esteja armazenada na posição seguinte (instruções de desvio)
- A instrução é armazenada no registrador de instrução (IR)
- O processador interpreta a instrução



Computador

- A instrução buscada na memória e carregada no IR contem bits que especificam a ação que o processador deve executar após a interpretação da instrução. Geralmente as ações são classificadas em quatro categorias:
  - Processador-memória: transferência de dados do processador para a memória ou da memória para o processador.
  - Processador-E/S: transferência de dados entre o processador e um dispositivo periférico.
  - Processamento de dados: execução de operações aritméticas ou lógicas sobre os dados.
  - Controle: alteração da sequência de execução de instruções

Computador Hipotético

- Registradores:
  - Contador de programa (PC) = endereço da próxima instrução
  - Registrador de instrução (IR) = instrução que está sendo executada
  - Acumulador (AC) = armazenamento temporário de dados

Computador Hipotético

- Códigos de operações:
  - 0001 = carregar AC a partir do endereço de memória especificado:

$$(AC) \leftarrow (mem)$$

  - 0010 = armazenar o valor contido em AC no endereço de memória especificado:

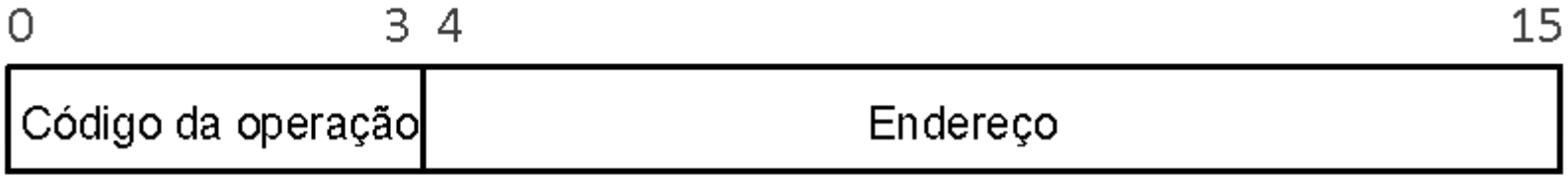
$$(mem) \leftarrow (AC)$$

  - 0101 = acrescentar ao valor contido em AC o valor contido no endereço de memória especificado:

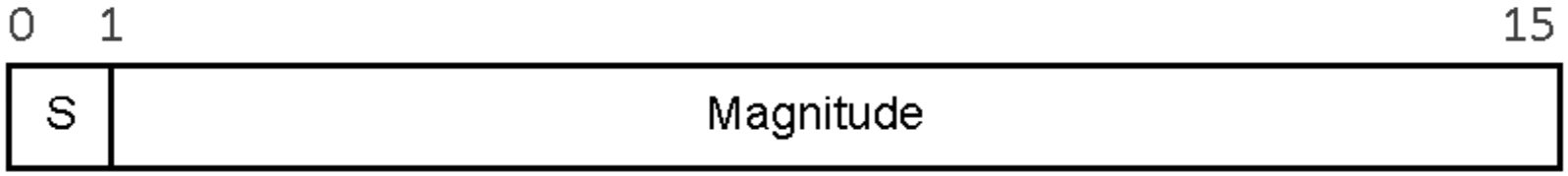
$$(AC) \leftarrow (AC) + (mem)$$

Computador Hipotético

Formato de instruções

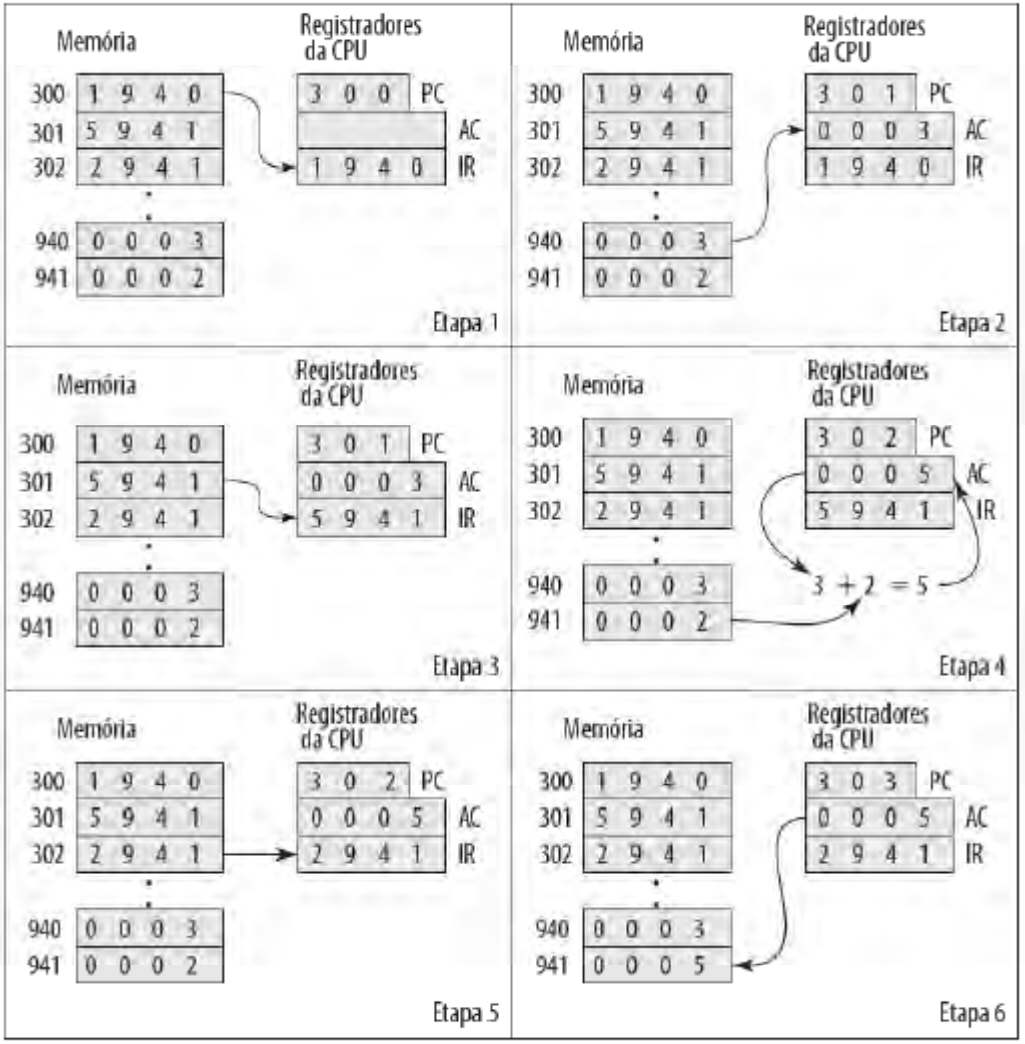


Formato de números inteiros



Computador Hipotético

- Soma de dois números
  - 0001 - (AC) ← (mem)
  - 0010 - (mem) ← (AC)
  - 0101 - (AC) ← (AC) +(mem)



Assembly

Assembly

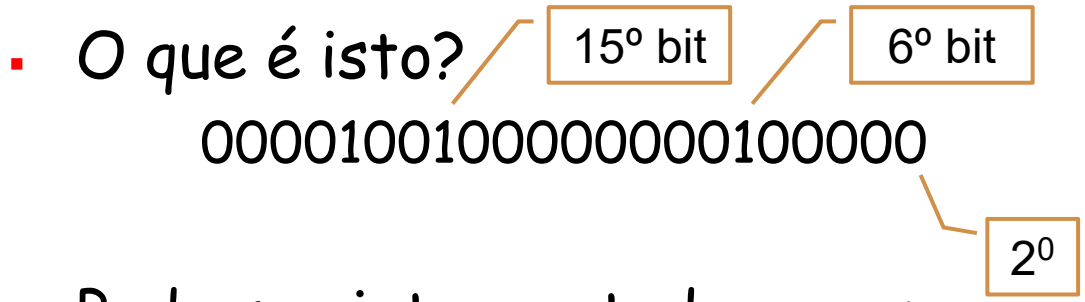
- Os números binários são base da teoria computacional
  - 100011000001 (bits): "Linguagem" do computador
  - 1. Primórdios: uso da linguagem nativa em binário.
  - 2. Linguagem de Montagem (Assembly)
    - Montador (Assembler): traduz uma versão simbólica das instruções para sua representação binária na arquitetura
    - add A, B -> montador -> 100011000001
  - 3. Linguagem de Programação de alto-nível
    - Compilador: traduz instruções de alto-nível para instruções binárias diretamente ou via um montador
    - A + B -> compilador -> add A, B -> montador -> 100011000001

Assembly

- Palavras (= conjunto de bits) -> podem ter diferentes significados agrupados em: dados e instruções
- Instruções: Contêm as informações que o computador necessita para executar as várias operações
- Cada máquina possui um conjunto de instruções (coleção completa de instruções que será entendida pela CPU)

Assembly

- O que é isto?



- Pode ser interpretado como:

$2^5 + 2^{14} + 2^{17}$

$= 32 + 16.384 + 13.1072$

$= 147.488$

Assembly

- O que é isto?

00000001000000010000001000001000000000100000
00000001000000110000010000001001000000100000
00000010000010000000100100001010000000100010

- Melhorando...

00000001 00000001 00000010 00001000 0000000 00100000
00000001 00000011 00000100 00001001 0000000 00100000
00000010 00001000 00001001 00000101 0000000 00100010

Assembly

- Melhor Assim?

  1 1 2 8 0 32

  1 3 4 9 0 32

  2 8 9 5 0 34

- Melhorando...

  add $8, $1, $2

  add $9, $3, $4

  sub $5, $8, $9

Assembly

- Melhor Assim?

  $8 = $1 + $2

  $9 = $3 + $4

  $5 = $8 - $9

- Melhorando...

  u = a + b

  v = c + d

  x = u - v

Assembly

- Claro agora?

  x = (a+b) - (c+d)

Assembly

- Linguagem de Montagem (Assembly): facilita a programação pois é uma abstração da linguagem de máquina.

- Nome simbólico, ou mnemônico, para cada instrução.
  Resultado: programa simbólico

- Ainda exige conhecimento do hardware por parte do programador

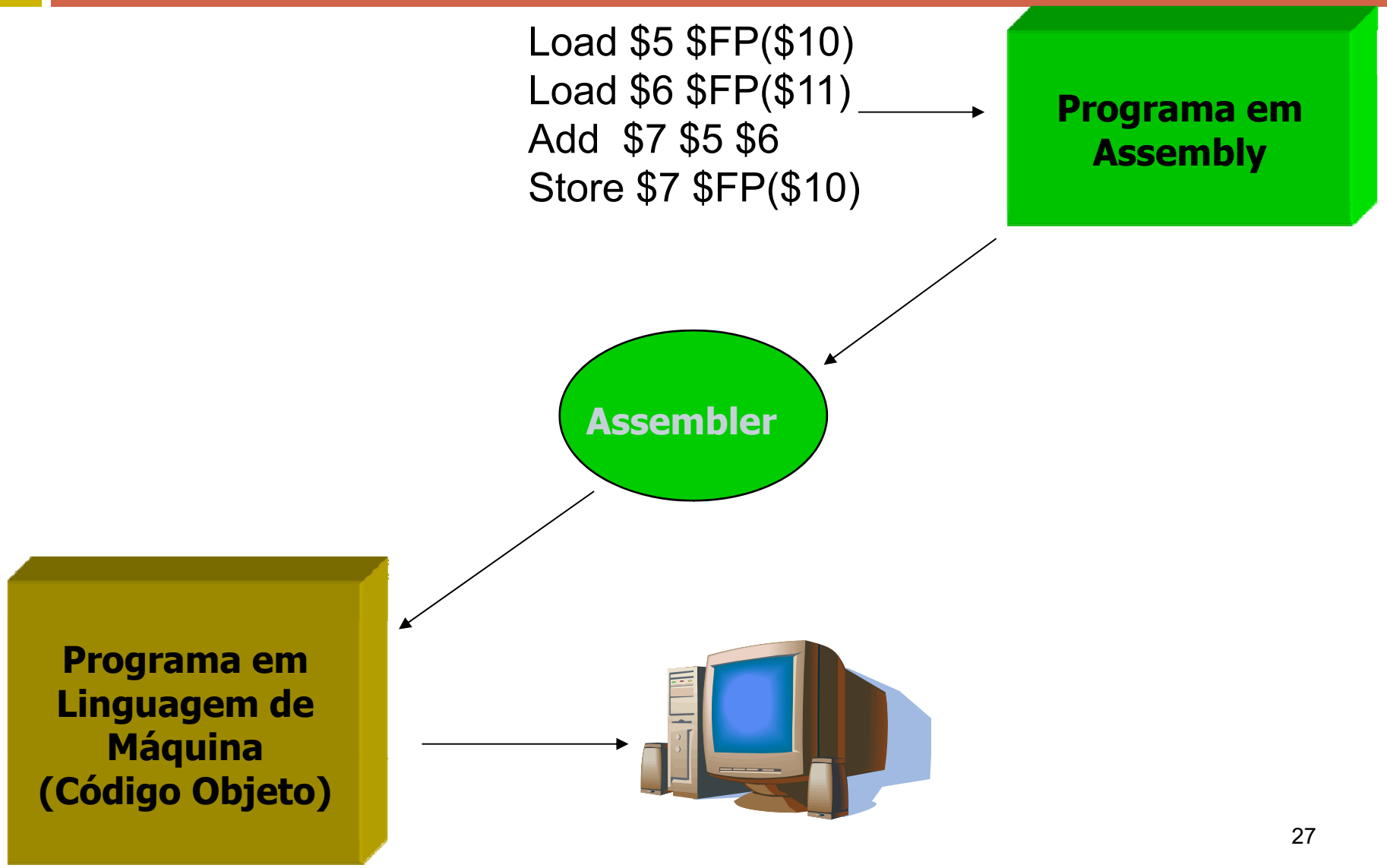
Assembly

Endereço	Conteúdo			
101	0010	0010	0000	0001
102	0001	0010	0000	0010
103	0001	0010	0000	0011
104	0011	0010	0000	0100
201	0000	0000	0000	0010
202	0000	0000	0000	0011
203	0000	0000	0000	0100
204	0000	0000	0000	0000

(a) Programa binário

Fonte: Arquitetura e Organização de Computadores - 5ª Ed. Stallings, William

Assembly


Load $5 $FP($10)
Load $6 $FP($11)
Add $7 $5 $6
Store $7 $FP($10)
Programa em Assembly
Assembler
Programa em Linguagem de Máquina (Código Objeto)

Assembler

- Montador (Assembler): Traduz programas escritos em linguagem de montagem para linguagem de máquina. Esse programa não apenas deve fazer a tradução simbólica mas também associar um endereço de memória a cada endereço simbólico.

- O desenvolvimento de linguagens de montagem constituiu um grande marco na evolução da tecnologia de computadores. Foi o primeiro passo para o desenvolvimento das linguagens de alto nível usadas atualmente.

Linguagens de Alto Nível

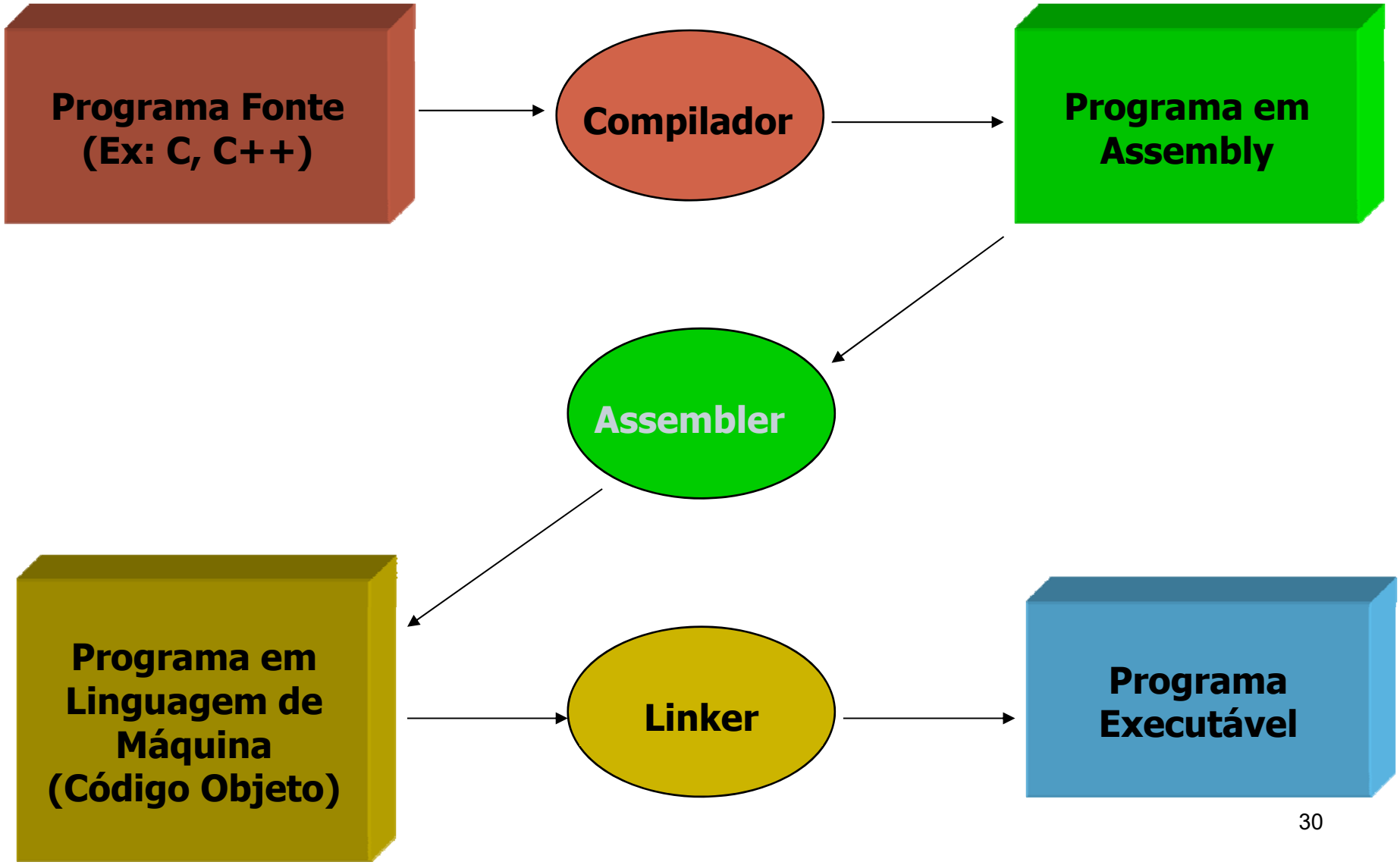
- Abstração de linguagem de máquina mais amigável do que linguagem assembly.

  Ex: C++, Java, Pearl, etc.

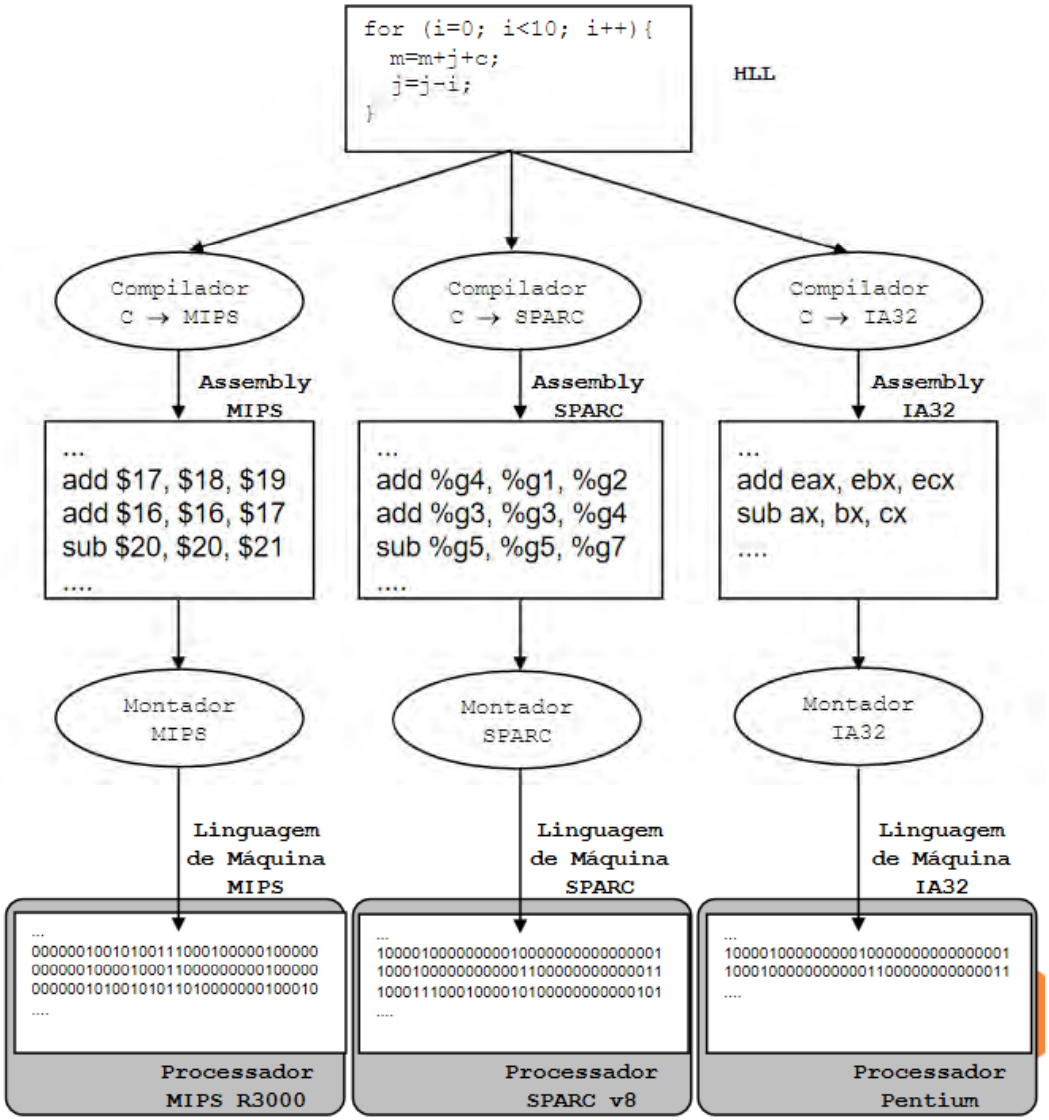
- Necessitam de um compilador para traduzir a linguagem de máquina em código assembly ou código de máquina.

- Conhecimento do hardware não é mais necessário para o programador, já que o compilador é responsável por isso.

Linguagens de Alto Nível



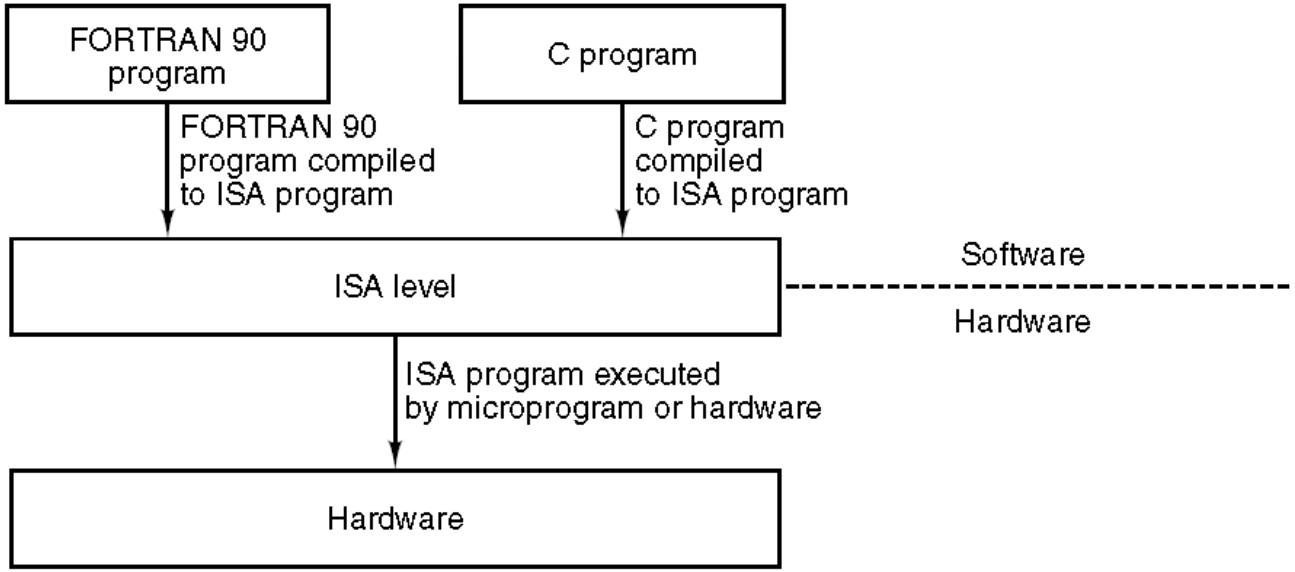
Linguagens de Alto Nível



ISA

ISA

- A operação de uma CPU é determinada pelas instruções que ela executa: instruções de máquina (operação elementar que o processador é capaz de realizar)
- Conjunto de instruções: limite de visão do projetista e do programador de computadores.



ISA

- Projetista: o conjunto de instruções de máquina fornece os requisitos funcionais para a CPU. Implementar uma CPU é uma tarefa que envolve, em grande parte, implementar o conjunto de instruções de máquina.

- Programador: o conjunto de instruções constitui o meio pelo qual o programador pode controlar a CPU.

  - Para utilizar o conjunto de instruções é necessário conhecer o conjunto de registradores da CPU, a estrutura de memória, os tipos de dados disponíveis diretamente na máquina e o funcionamento da ULA, ou seja: a arquitetura do conjunto de instruções

ISA

ISA - Instruction Set Architecture

Arquitetura do conjunto de instruções

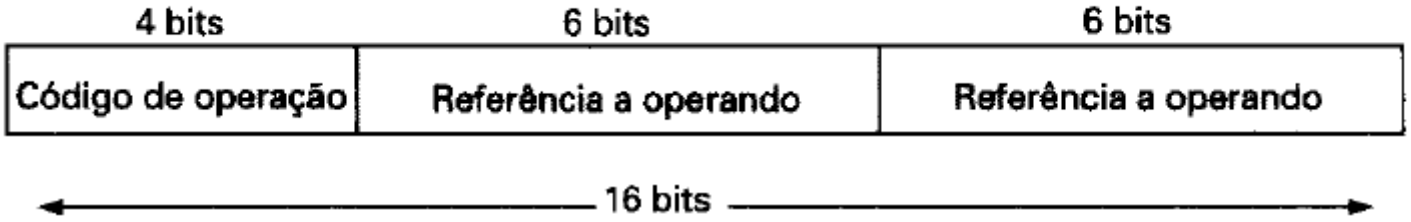
ISA

- Elementos essenciais de uma instrução:
  - Código de operação, que especifica a operação a ser realizada,
  - Referências aos operandos de origem e de destino, que especificam os endereços dos dados de entrada e de saída da operação.
  - Endereço da próxima instrução, que normalmente é implícito.

- Os dados necessários ou gerados pelas instruções podem ser lidos ou escritos das seguinte áreas:
  - Memória principal
  - Registradores da CPU
  - Dispositivos de I/O

ISA

- Representação das Instruções
  - Cada instrução é representada como uma sequencia de Bits
  - A instrução é dividida em pequenos campos. Cada campo representa um elemento da instrução
  - Durante a execução de uma instrução, a instrução é armazenada no IR (Instruction Register) da CPU.
  - A CPU encarrega-se de interpretar os bits da instrução



ISA

- Representação de instruções
  - Código de operação são representados por abreviações (mnemônicos) que é uma representação simbólica para instruções. Exemplos:

ADD	Adição
SUB	Subtração
MPY	Multiplicação
DIV	Divisão
LOAD	Carregar dados da memória
STOR	Armazenar dados na memória

  - Operandos também são representados de maneira simbólica

```
ADD  R, Y
```

ISA

- Tipos de Instruções
  - Processamento de dados: instruções aritméticas e lógicas
  - Armazenamento de dados: instruções de memória
  - Movimentação de dados: instruções de E/S
  - Controle: instruções de teste e de desvio

ISA

- Número de Endereços
  - Instruções mais comuns teriam obrigatoriedade de ter 4 endereços de referência:
    - 2 operandos, 1 resultado e o endereço da instrução seguinte
    - Na prática, 4 endereços é uma situação extremamente rara
    - Muitas CPUs possuem 1,2, ou 3 endereços na instrução, com o endereço da próxima instrução estando explícito através do PC

ISA

- Número de Endereços

Número de endereços	Representação simbólica	Interpretação
3	OP A, B, C	A ← B OP C
2	OP A, B	A ← A OP B
1	OP A	AC ← AC OP A
0	OP	T ← (T–1) OP T

AC = acumulador
T = topo da pilha
A, B, C = registrador ou posição de memória

ISA

- Número de Endereços

Instrução		Comentário
SUB	Y, A, B	Y ← A − B
MPY	T, D, E	T ← D × E
ADD	T, T, C	T ← T + C
DIV	Y, Y, T	Y ← Y ÷ T

(a) Instruções com três endereços

Instrução		Comentário
MOVE	Y, A	Y ← A
SUB	Y, B	Y ← Y − B
MOVE	T, D	T ← D
MPY	T, E	T ← T × E
ADD	T, C	T ← T + C
DIV	Y, T	Y ← Y ÷ T

(b) Instruções com dois endereços

Instrução		Comentário
LOAD	D	AC ← D
MPY	E	AC ← AC × E
ADD	C	AC ← AC + C
STOR	Y	Y ← AC
LOAD	A	AC ← A
SUB	B	AC ← AC − B
DIV	Y	AC ← AC ÷ Y
STOR	Y	Y ← AC

(c) Instruções com um endereço

ISA

- Projeto do Conjunto de Instruções
  - Aspecto mais interessante e mais analisados do projeto de computadores.
  - Muito complexo, pois afeta diversos aspectos do sistema.
  - Define muitas das funções desempenhadas pela CPU tendo efeito significativo sobre a implementação da CPU.
  - Como o conjunto de instruções constitui o meio pelo qual o programador pode controlar a CPU portanto, ao projetar um conjunto de instruções é necessário considerar as necessidades do programador

ISA

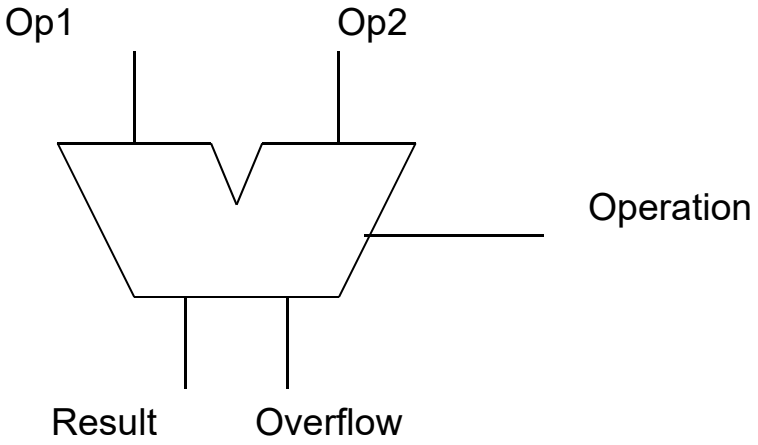
- Projeto do Conjunto de Instruções
  - Operações: quantas e quais são as operações que devem ser fornecidas e quão complexas elas podem ser.
  - Tipos de dados: quais os tipos de dados sobre os quais as operações são efetuadas.
  - Formatos de instrução: qual o tamanho das instruções, o número de endereços por instrução, o tamanho dos vários campos etc.
  - Registradores: qual o número de registradores da CPU que podem ser usados pelas instruções e qual o propósito de cada registrador.
  - Endereçamento: como o endereço de um operando pode ser especificado.

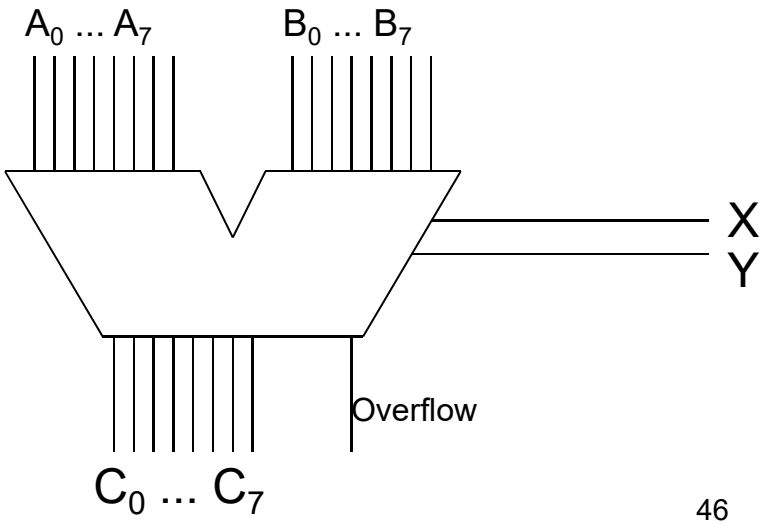
ISA

- Uma forma de se categorizar possíveis tipos de ISAs é de acordo com o armazenamento dos operandos das instruções:
  - Arquitetura baseada em Pilha (stack)
    - Implicitamente, os operandos estão no topo de uma pilha
  - Arquitetura baseada em Acumulador (accumulator)
    - Implicitamente, um operando é o acumulador, e o outro é referenciado explicitamente
  - Arquitetura baseada em Registradores (register)
    - Todos os operandos são referenciados explicitamente (memória e/ou registrador)

ISA

- ULA – Unidade Lógica Aritmética (ALU)



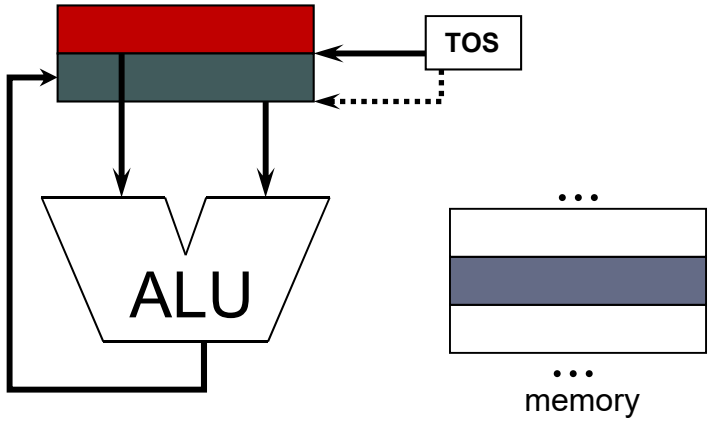


ISA

- Arquitetura baseada em Pilha
  - O hardware implementa a pilha, reservando uma região na memória que irá armazenar a pilha
  - Usa um registrador p/ armazenar o endereço do topo da pilha (TOS)
  - A ISA fornece instruções PUSH e POP, além de outras convencionais como ADD, SUB, etc.
  - Ex: Os operandos da instrução ADD são implicitamente os dois elementos no topo da pilha. ADD faz POP dos dois elementos no topo da pilha, soma-os, e faz PUSH do resultado no topo da pilha.

ISA

- Arquitetura baseada em Pilha




```
PUSH A    // load the data at @ A at the top
PUSH B    // load the data at @ B at the top
ADD       // pop the two two elements,
             add them, push the result
POP C     // store the top at @ C
```

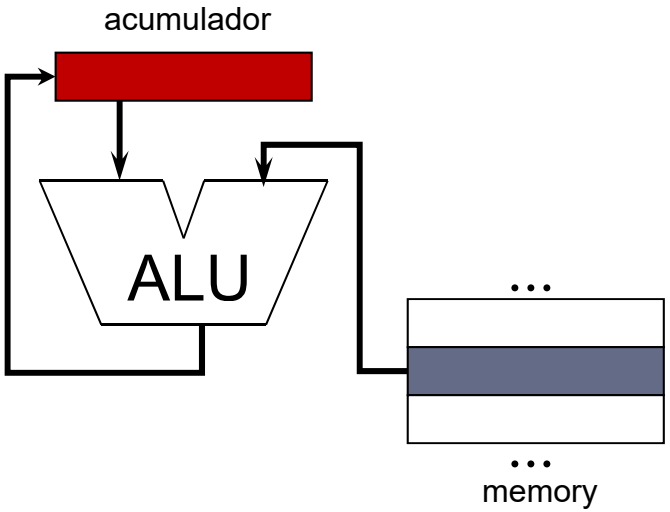
Toda a vez que a pilha é modificada, o registrador TOS é atualizado automaticamente

ISA

- Arquitetura baseada em Acumulador
  - O hardware implementa o acumulador, que é um registrador
  - A ISA fornece instruções LOAD e STORE instruction, além de ADD, SUB, etc.
  - LOAD e STORE fazem a transferência de dados entre o acumulador e a memória
  - Ex: Na instrução ADD, um operando é implicitamente o conteúdo do acumulador, e o outro um conteúdo da memória. O resultado é armazenado no acumulador.

ISA

- Arquitetura baseada em Acumulador



LOAD A **// load the data at @ A in the acc.**
ADD B **// add B to the acc.**
STORE C **// store the content of the**
// acc. at @ C

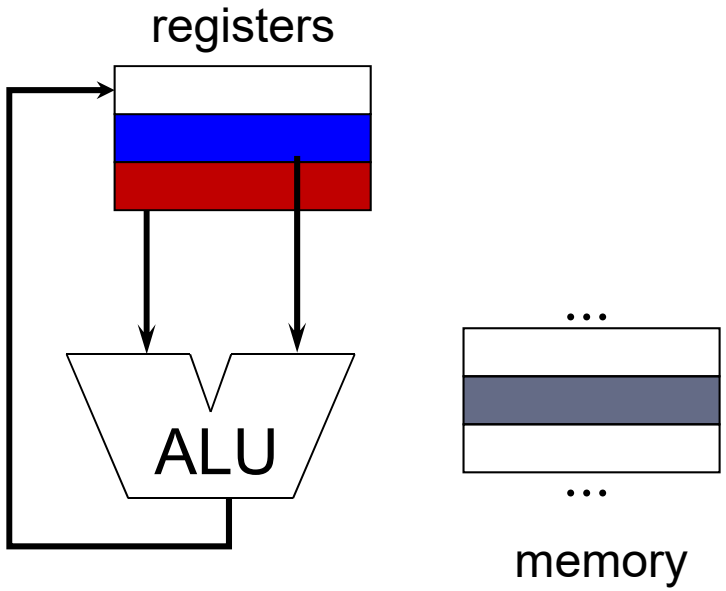
Utiliza menos instruções do que a arquitetura baseada em pilha

ISA

- Arquitetura baseada em Registrador
  - Todos os operandos de uma instrução são explícitos, podendo ser: registradores ou células de memória RAM.
  - Opção #1: register-register ISAs
    - Instrução pode ter apenas operandos em registradores (ex: MIPS)
    - Também conhecidas como load-store architectures
  - Opção #2: register-memory ISAs
    - Instrução pode ter algum operando na memória (ex: x86)
  - Ambas as opções são chamadas General Purpose Register (GPR) ISAs

ISA

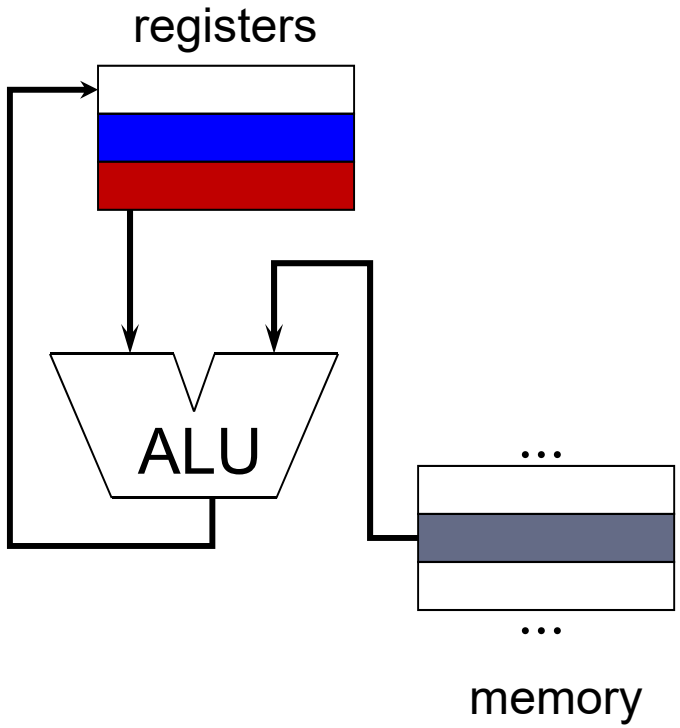
- Arquitetura baseada em Registrador
  - Register-Register



LOAD R1, A
LOAD R2, B
ADD R3, R1, R2
STORE R3, C

ISA

- Arquitetura baseada em Registrador
  - Register-Memory



LOAD R1, A
ADD R3, R1, B
STORE R3, C

ISA

- Arquitetura baseada em Registrador
  - Desde os anos 80, praticamente todo novo projeto de processador é baseado em registradores (GPR ISA).
  - Registradores são bem mais rápidos que a memória, e se tornaram mais baratos de implementar
  - Pilha e acumuladores dificultam a tarefa do compilador, impedindo a geração de código mais eficiente (otimizações).
    - Ex: uso de registradores temporários, reduzindo o tráfego entre memória <- > pilha/acumulador
    - (A*B) - (B*C) - (A*D)

MIPS operands

Name	Example	Comments
32 registers	$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register $zero always equals 0, and register $at is reserved by the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Three register operands
	subtract	sub $s1,$s2,$s3	$s1 = $s2 – $s3	Three register operands
	add immediate	addi $s1,$s2,20	$s1 = $s2 + 20	Used to add constants
Data transfer	load word	lw $s1,20($s2)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	and	and $s1,$s2,$s3	$s1 = $s2 & $s3	Three reg. operands; bit-by-bit AND
	or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Three reg. operands; bit-by-bit OR
	nor	nor $s1,$s2,$s3	$s1 = ~ ($s2 | $s3)	Three reg. operands; bit-by-bit NOR
	and immediate	andi $s1,$s2,20	$s1 = $s2 & 20	Bit-by-bit AND reg with constant
	or immediate	ori $s1,$s2,20	$s1 = $s2 | 20	Bit-by-bit OR reg with constant
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Shift left by constant
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Shift right by constant
Conditional branch	branch on equal	beq $s1,$s2,25	if ($s1 == $s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne $s1,$s2,25	if ($s1!= $s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set on less than unsigned	sltu $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than unsigned
	set less than immediate	slti $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant
	set less than immediate unsigned	sltiu $s1,$s2,20	if ($s2 < 20) $s1 = 1; else $s1 = 0	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC + 4; go to 10000	For procedure call

ISA

- Quantos Registradores ?
  - ISAs mais modernas tendem a ter um número maior de registradores visíveis
    - 8, 16, 32, 64, 128
  - Isso facilita o trabalho do compilador, resultando em código mais rápido e eficiente

ISA

- Quantos operandos por Instrução ?
  - GPR ISAs tem duas características básicas:
    1. Se a ALU tem 2 ou 3 operandos
       - 3 operandos: o caso mais intuitivo:

         ADD $5, $4, $3 (Reg5= Reg4 + Reg3)
       - 2 operandos: um deles é tanto entrada como saída

         ADD $4, $3 (Reg4= Reg4 + Reg3)
    2. O número de operandos que pode ser um endereço de memória

ISA

Número máximo de operandos	Número de operandos na memória	Tipo de Arquitetura	Exemplos de Arquiteturas Comerciais
3	0	Register-register	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, Trimedia TM5200
2	1	Register-memory	IBM 360,370, Intel 80x86, Motorola 68000, TI TMS320C54x
2	2	Memory-memory	VAX
3	3	Memory-memory	VAX

Iremos nos concentrar na configuração (3,0), ou seja, 3 operandos em registradores, que é a opção mais moderna.

ISA: Características Buscadas

- Uma boa ISA resulta em:
  - Simplicidade do hardware necessário para implementá-la
  - Clareza da sua funcionalidade e aplicação
  - Velocidade de processamento