Chapter 2: Instructions: Language of the Computer

2.5. Representing Instructions in the Computer

Arquitetura e Organização de Computadores

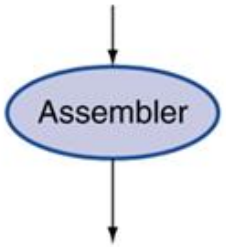
Introdução: 3 classes de instruções

- Aritméticas e lógicas:
  - **`add`**, **`sub`**, **`addi`**
  - **`and`**, **`or`**, **`shift left`**, **`shift right`**
- Carregar da memória e armazenar na memória:
  - **`lw`**, **`sw`**
  - **`ld`**, **`sd`**
- Transferência de controle (alteram a seqüência de execução das instruções):
  - Ramificação condicional: **`bne`**, **`beq`**
  - "Pulo" não condicional: **`j`**
  - Chamada e retorno de procedimentos: **`jal`**, **`jar`**

Problema: homem x computador

- Como “traduzir” a forma como os humanos escrevem as instruções (assembly) na forma com a qual os computadores entendem essas instruções?

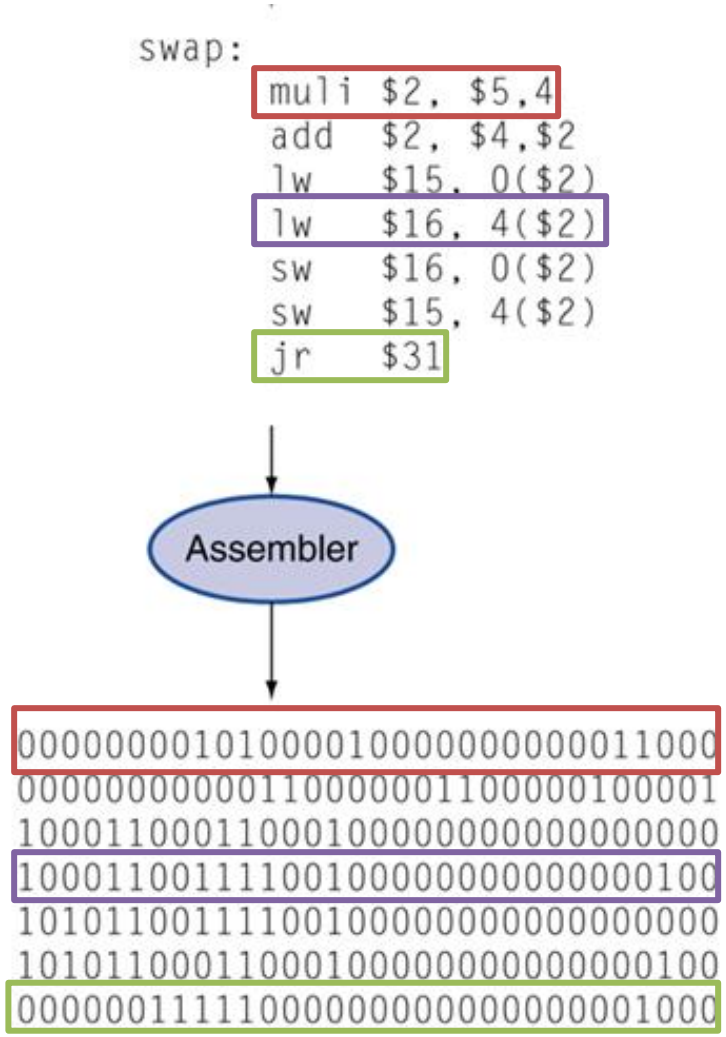
```
swap:
        muli $2, $5,4
        add  $2, $4,$2
        lw   $15, 0($2)
        lw   $16, 4($2)
        sw   $16, 0($2)
        sw   $15, 4($2)
        jr   $31
```




```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

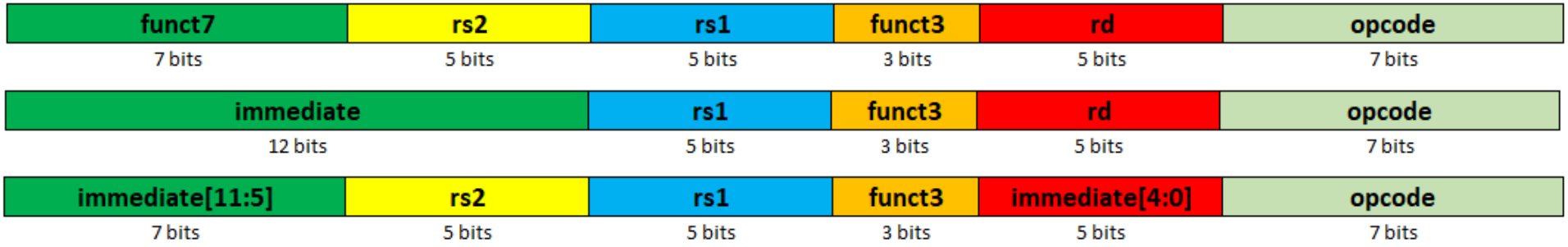
Representando instruções

- As instruções são codificadas em binário:
  - Uma instrução em **linguagem assembly** é "traduzida" para uma versão binária dessa mesma instrução, e essa versão binária é chamada de instrução em **linguagem de máquina**
  - **Linguagem de máquina**: é a representação binária de instruções assembly, usada na "comunicação" com o computador
  - **Código de máquina**: uma seqüência de instruções em linguagem de máquina



Representando instruções

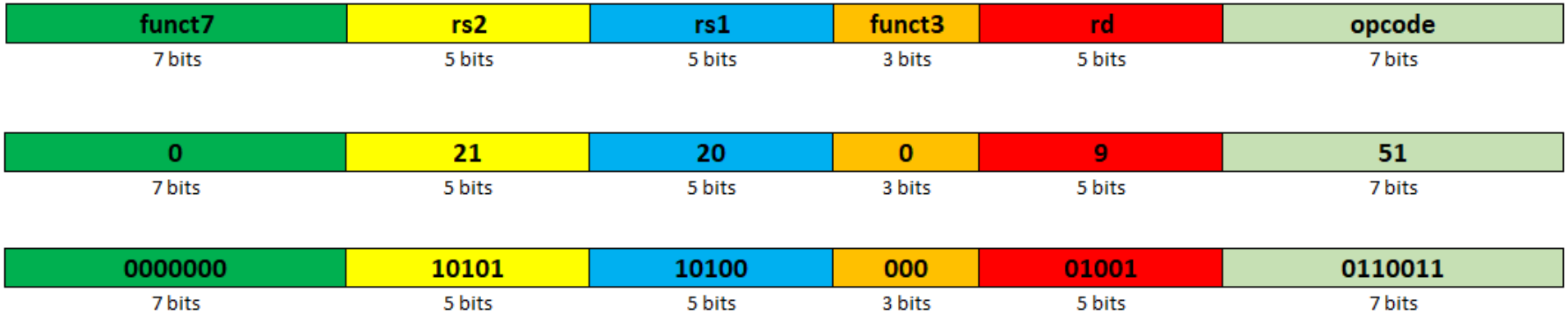
- No RISC-V:
  - Cada instrução em assembly é traduzida como uma instrução em linguagem de máquina com 32 bits (palavra de instrução)
  - Existem **diferentes formatos** para representar as instruções, dependendo da "classe" de instrução, por exemplo:
    - Formato-R, Formato-I, Formato-S, ...



  - Cada instrução de 32 bits é dividida em **campos**. Três desses campos indicam o qual a instrução a ser executada
    - (verde, laranja e verde claro)
  - **Formato da instrução**: o "layout", a divisão de campos de cada instrução

Representando instruções: Formato-R

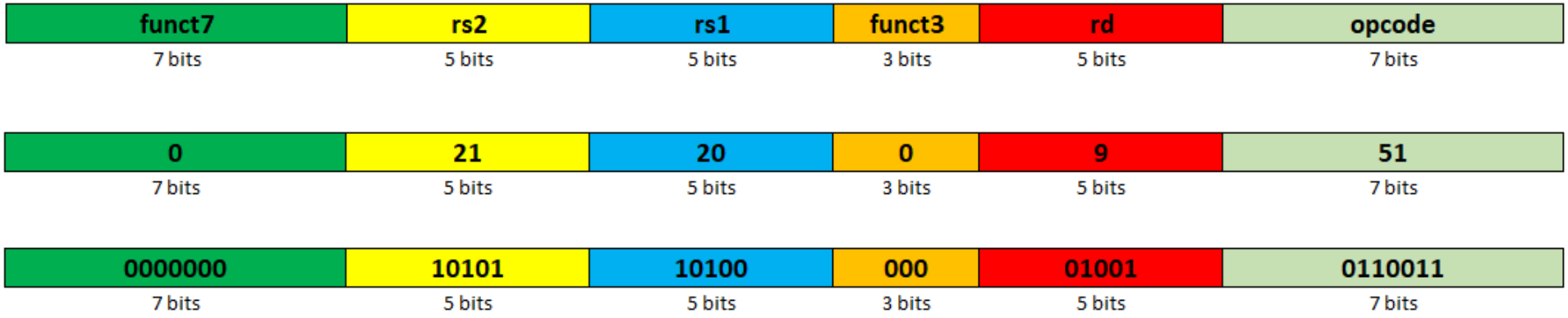
- Exemplo:
  - `add x9, x20, x21`



- Campos no RISC-V:
  - **opcode**: operação e formato da instrução
  - **rd**: registrador de destino
  - **funct3**: opcode adicional
  - **rs1**: primeiro registrador de origem
  - **rs2**: segundo registrador de origem
  - **funct7**: opcode adicional

Representando instruções: Formato-R

- Note que a instrução a seguir, em assembly:
  - `add x9, x20, x21`



- Corresponde à instrução abaixo, em linguagem de máquina:
  - $0000000101011010000001001011001 1_2$
  - $015A04B3_{16}$
- Uso de hexadecimal facilita a escrita, mas tudo é em binário

Adendo: hexadecimal

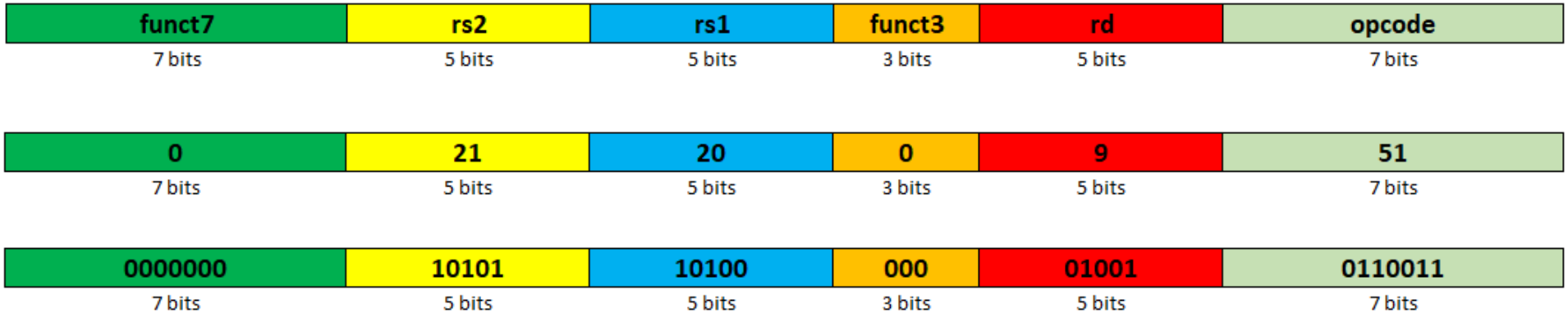
- Usar base 16 facilita representar números binários grandes
- Cada dígito hexadecimal corresponde a 4 dígitos binários
- A conversão é direta, em ambos os sentidos:
  - Hexadecimal para binário
  - Binário para hexadecimal

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

- Exemplo: **`eca86420`** corresponde a que binário?

Representando instruções: Formato-R

- **`rs1`**, **`rs2`** e **`rd`** são fáceis de entender
- Quem define quais são os opcodes?



– O **opcode**, **funct3** e **funct7** são estabelecidos pelo ISA! Na documentação do RISC-V temos, por exemplo:

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Representando instruções: Formato-I

- Servem para instruções que contém um “número” (“immediate”), um registrador de origem e um registrador de destino, por exemplo:
  - `addi x22, x22, 4`
  - `lw   x9, 64(x22)`
- Atenção:
  - **`rs1`** é o registrador de origem OU o endereço base
  - **`immediate`** é o número OU o offset do endereço base
    - São 12 bits, em complemento de dois
  - Instruções de Format-I não servem para salvar dados na memória porque o destino precisa ser uma posição de memória, não um registrador.

Representando instruções: Formato-I

- Servem para instruções que contém um “número” (“immediate”), um registrador de origem e um registrador de destino, por exemplo:
  - `addi x22, x22, 4`
  - `lw x9, 64(x22)`
- Como seriam essas instruções?
  - Verifique os opcodes corretos no RISC-V e escreva a instrução em binário ou hexadecimal

Representando instruções: Formato-S

- Dois registradores de origem, nenhum registrador de destino
- Usadas somente para armazenar instruções/dados:
  - `sw x9, 96(x22)`

- Atenção:
  - rs1: endereço base
  - rs2: valor a ser armazenado na memória
  - immediate: offset
    - É dividido para manter os campos rs1 e rs2 sempre no mesmo local que as instruções no formato R e I

Mais exemplos

Instruction	Format	funct7	rs2	rs1	funct3	rd	opcode
`add` (add)	R	0000000	reg	reg	000	reg	0110011
`sub` (sub)	R	0100000	reg	reg	000	reg	0110011

Instruction	Format	immediate	rs1	funct3	rd	opcode
`addi` (add immediate)	I	constant	reg	000	reg	0010011
`lw` (load word)	I	address	reg	010	reg	0000011

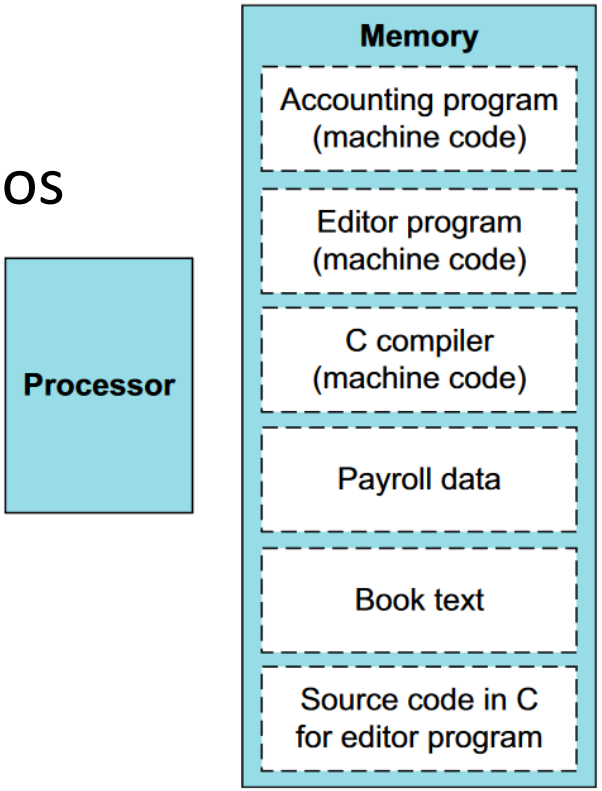
Instruction	Format	immed -iate	rs2	rs1	funct3	immed -iate	opcode
`sw` (store word)	S	address	reg	reg	010	address	0100011

Mais exemplos

R-type Instructions	funct7	rs2	rs1	funct3	rd	opcode	Example
add (add)	0000000	00011	00010	000	00001	0110011	add x1, x2, x3
sub (sub)	0100000	00011	00010	000	00001	0110011	sub x1, x2, x3
I-type Instructions	**immediate**		**rs1**	**funct3**	**rd**	**opcode**	**Example**
addi (add immediate)	001111101000		00010	000	00001	0010011	addi x1, x2, 1000
lw (load word)	001111101000		00010	010	00001	0000011	lw x1, 1000 (x2)
S-type Instructions	**immed -iate**	**rs2**	**rs1**	**funct3**	**immed -iate**	**opcode**	**Example**
sw (store word)	0011111	00001	00010	010	01000	0100011	sw x1, 1000(x2)

Conceito: Programa Armazenado

- Instruções são representadas em binário, assim como qualquer outro dado
- Instruções e dados são armazenados na memória
- Programas são armazenados na memória para serem lidos ou escritos, assim como os dados
- Programas podem operar em programas
  - Compiladores, linkers, etc.
- Compatibilidade binária permite que programas compilados possam funcionar em diferentes computadores
  - ISAs padronizados



Sua vez!

- Questão 01: um programador escreveu o seguinte código fonte (A é um array de inteiros; h é uma variável que contém um inteiro).

```
A[30] = h + A[30] + 1;
```

Considerando que o registrador **x10** contém o endereço base do array **A**, que o registrador **x21** contém o valor da variável **h**, e que você utilizará o registrador **x9** para armazenamento temporário, faça:

a) Escreva o código assembly correspondente;

b) Escreva todas as instruções necessárias, em linguagem de máquina

Hora de Esfriar a Cabeça!