Linguagem de máquina e de montagem

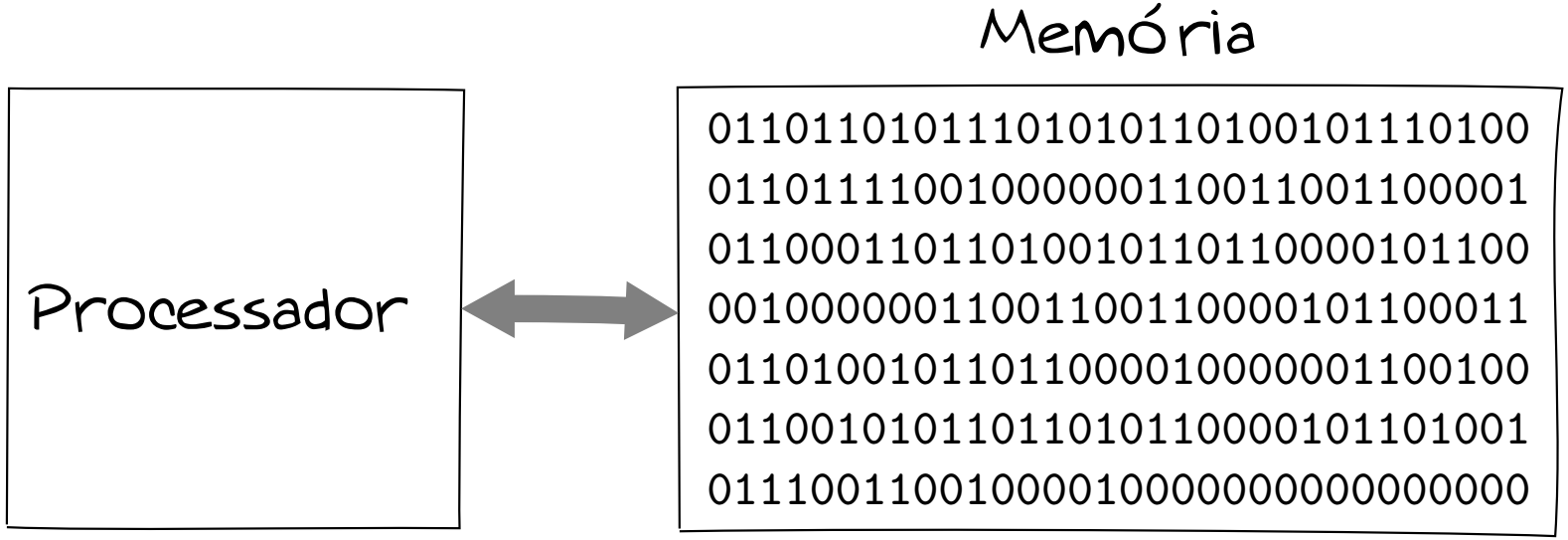
Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é linguagem de máquina?
  - É a representação em formato binário das instruções que o processador é capaz de entender e executar



Introdução

- O que é linguagem de montagem (*assembly*)?
  - É uma linguagem de programação de baixo nível que utiliza mnemônicos e dados em formatos decimal ou hexadecimal para descrição das instruções

Linguagem de montagem

```
l32 r1, [0x40]
cmpi r1, 0
beq 2
subi r1, r1, 1
bun -4
s32 [0x40], r1
int 0
```

Memória

```
01101101011101010110100101110100
01101111001000000110011001100001
01100011011010010110110000101100
00100000011001100110000101100011
01101001011011000010000001100100
01100101011011010110000101101001
01110011001000010000000000000000
```

Introdução

- O que é linguagem de montagem (*assembly*)?
  - É uma linguagem de programação de baixo nível que utiliza mnemônicos e dados em formatos decimal ou hexadecimal para descrição das instruções

Linguagem de montagem

```
l32 r1, [0x40]
cmpi r1, 0
beq 2
subi r1, r1, 1
bun -4
s32 [0x40], r1
int 0
```

Memória

```
01101101011101010110100101110100
01101111001000000110011001100001
01100011011010010110110000101100
00100000011001100110000101100011
01101001011011000010000001100100
01100101011011010110000101101001
01110011001000010000000000000000
```

O montador (*assembler*) é responsável por traduzir o código de montagem para linguagem de máquina

Introdução

- Representação binária, decimal e hexadecimal
  - Normalmente os seres humanos trabalham com números representados em base decimal (10 dígitos)
  - Os dados e as instruções nos computadores são representadas em formato binário (2 dígitos)

Introdução

- Representação binária, decimal e hexadecimal
  - Normalmente os seres humanos trabalham com números representados em base decimal (10 dígitos)
  - Os dados e as instruções nos computadores são representadas em formato binário (2 dígitos)

$$
\begin{aligned}
123_{10} &= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 \\
&= 1 \times 2^6 + 59 \\
&= 1 \times 2^6 + 1 \times 2^5 + 27 \\
&= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 11 \\
&= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 3 \\
&= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 \\
&= 01111011_2
\end{aligned}
$$

Introdução

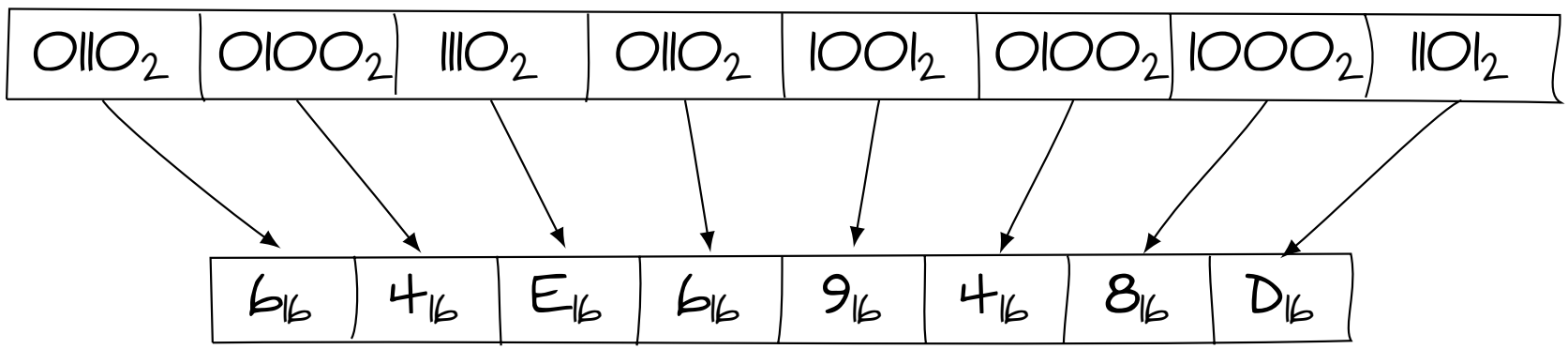
- Representação binária, decimal e hexadecimal
  - Normalmente os seres humanos trabalham com números representados em base decimal (10 dígitos)
  - Os dados e as instruções nos computadores são representadas em formato binário (2 dígitos)

$$
\begin{aligned}
123_{10} &= 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 \\
&= 1 \times 2^6 + 59 \\
&= 1 \times 2^6 + 1 \times 2^5 + 27 \\
&= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 11 \\
&= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 3 \\
&= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 \\
&= 01111011_2
\end{aligned}
$$

A notação posicional pode ser utilizada em qualquer base numérica

Introdução

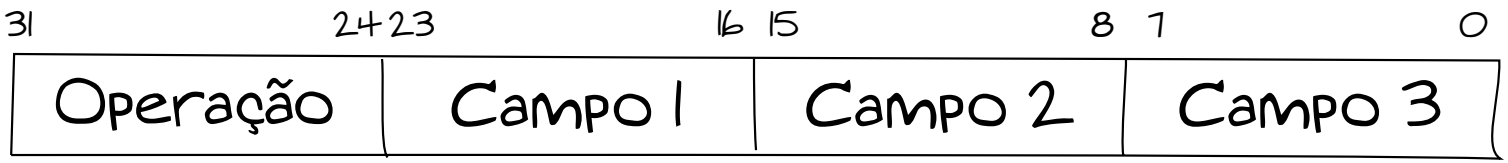
- Representação binária, decimal e hexadecimal
  - O formato hexadecimal permite codificar diretamente números binários agrupados em *nibbles*, reduzindo significativamente a quantidade de dígitos



$$01100100111001101001010010001101_2 = 64E6948D_{16}$$

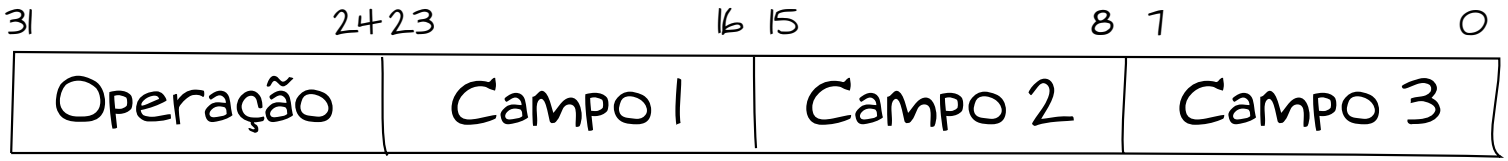
Introdução

- Como entender, organizar e traduzir as instruções para linguagem de máquina?
  - Todas as informações são números em formato binário
  - É preciso fazer as conversões numéricas e atender ao formatos definidos na especificação (*datasheet*)



Introdução

- Como entender, organizar e traduzir as instruções para linguagem de máquina?
  - Todas as informações são números em formato binário
  - É preciso fazer as conversões numéricas e atender ao formatos definidos na especificação (*datasheet*)



Regularidade <-> Simplicidade

Introdução

- Paradigmas de computação
  - *Complex Instruction Set Computing* (CISC)
    - Focado na linguagem de programação, simplificando a tradução para linguagem de máquina e deixando a arquitetura do processador mais complexa
  - *Reduced Instruction Set Computing* (RISC)
    - Centrado no funcionamento do processador, simplificando a arquitetura de hardware e tornando o software mais complexo (incluindo o compilador)

Introdução

- Paradigmas de computação
  - *Complex Instruction Set Computing* (CISC)
    - Focado na linguagem de programação, simplificando a tradução para linguagem de máquina e deixando a arquitetura do processador mais complexa
    - Instruções com operandos em memória
  - *Reduced Instruction Set Computing* (RISC)
    - Centrado no funcionamento do processador, simplificando a arquitetura de hardware e tornando o software mais complexo (incluindo o compilador)
    - Arquitetura de carregamento-armazenamento

Introdução

- Paradigmas de computação
  - *Complex Instruction Set Computing* (CISC)
    - Focado na linguagem de programação, simplificando a tradução para linguagem de máquina e deixando a arquitetura do processador mais complexa
    - Instruções com operandos em memória
    - Maior densidade de código
  - *Reduced Instruction Set Computing* (RISC)
    - Centrado no funcionamento do processador, simplificando a arquitetura de hardware e tornando o software mais complexo (incluindo o compilador)
    - Arquitetura de carregamento-armazenamento
    - Menor densidade de código

Introdução

- Paradigmas de computação
  - *Complex Instruction Set Computing* (CISC)
    - Focado na linguagem de programação, simplificando a tradução para linguagem de máquina e deixando a arquitetura do processador mais complexa
    - Instruções com operandos em memória
    - Maior densidade de código
    - Ex: IBM, Intel x86, Z80, ...
  - *Reduced Instruction Set Computing* (RISC)
    - Centrado no funcionamento do processador, simplificando a arquitetura de hardware e tornando o software mais complexo (incluindo o compilador)
    - Arquitetura de carregamento-armazenamento
    - Menor densidade de código
    - Ex: ARM, MIPS, RISC-V, ...

Introdução

- Paradigmas de computação
  - *Complex Instruction Set Computing* (CISC)
    - Focado na linguagem de programação, simplificando a tradução para linguagem de máquina e deixando a arquitetura do processador mais complexa
    - Instruções com operandos em memória
    - Maior densidade de código
    - Ex: IBM, Intel x86, Z80, ...
  - *Reduced Instruction Set Computing* (RISC)
    - Centrado no funcionamento do processador, simplificando a arquitetura de hardware e tornando o software mais complexo (incluindo o compilador)
    - Arquitetura de carregamento-armazenamento
    - Menor densidade de código
    - Ex: ARM, MIPS, RISC-V, ...

Convergência

CISC $\rightleftarrows$ RISC

Introdução

- Por que escolher uma arquitetura RISC?
  - A formatação regular e simples das instruções permite a decodificação e execução mais eficiente das operações, reduzindo a área de silício necessária, assim como o custo unitário e o consumo de potência

Introdução

- Por que escolher uma arquitetura RISC?
  - A formatação regular e simples das instruções permite a decodificação e execução mais eficiente das operações, reduzindo a área de silício necessária, assim como o custo unitário e o consumo de potência
  - Na arquitetura de carregamento-armazenamento é reduzido o número de acessos à memória e a latência das operações que são realizadas exclusivamente em registradores, explorando os princípios de localidade espacial e temporal

Introdução

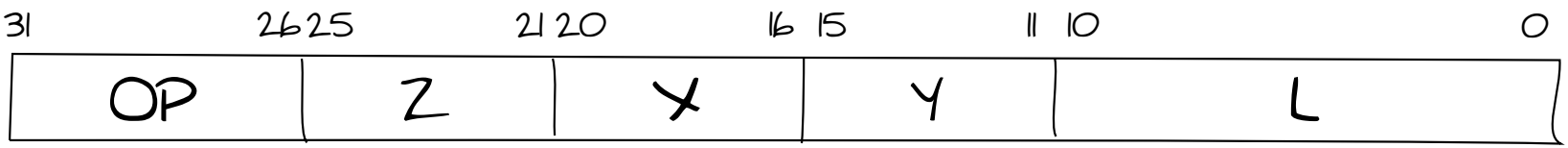
- Por que escolher uma arquitetura RISC?
  - A formatação regular e simples das instruções permite a decodificação e execução mais eficiente das operações, reduzindo a área de silício necessária, assim como o custo unitário e o consumo de potência
  - Na arquitetura de carregamento-armazenamento é reduzido o número de acessos à memória e a latência das operações que são realizadas exclusivamente em registradores, explorando os princípios de localidade espacial e temporal
  - Simplifica a implementação de técnicas para execução sobreposta (*pipeline*) ou paralela (superescalar) de instruções no processador

Linguagem de máquina

- Arquitetura Poxim
  - *Complexity-Reduced Instruction Set Processor* (CRISP)
  - Didática, hipotética e simples com 32 bits
  - Memória Von Neumann de 32 KiB
  - 3 formatos de instruções

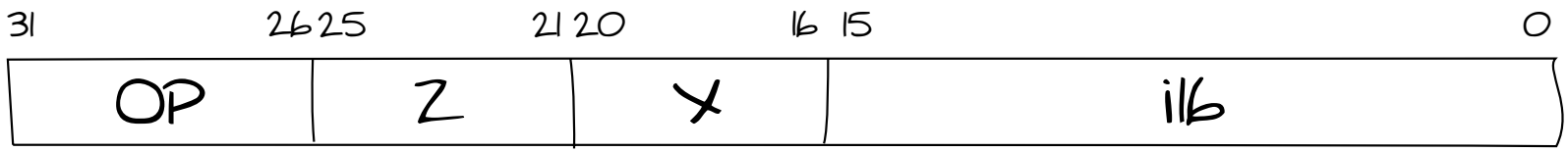
Linguagem de máquina

- Formato U (OP, Z, X, Y, L)
  - 6 bits para operação (OP)
  - 5 bits para operandos (Z, X, Y)
  - 11 bits para uso livre (L)



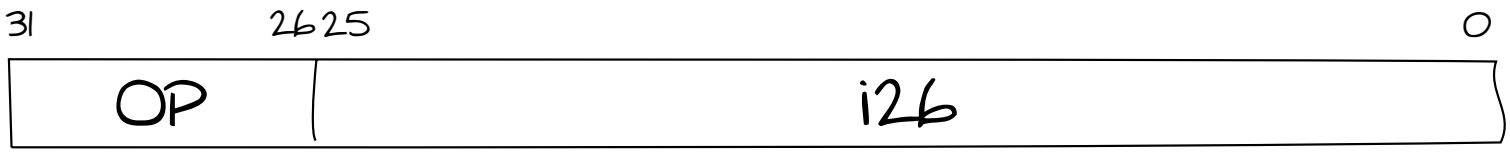
Linguagem de máquina

- Formato F (OP, Z, X, I16)
  - 6 bits para operação (OP)
  - 5 bits para operandos (Z, X)
  - 16 bits para imediato (I16)



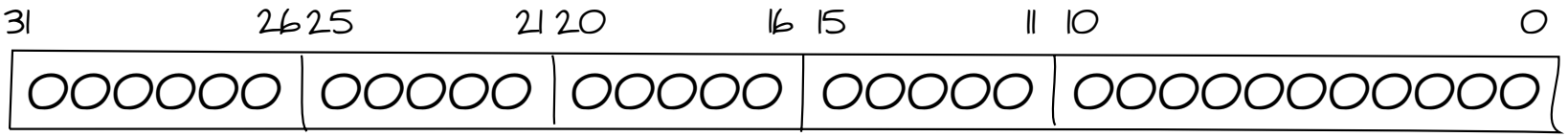
Linguagem de máquina

- Formato S (OP, I26)
  - 6 bits para operação (OP)
  - 26 bits para imediato (I26)



Linguagem de máquina

- Operação ociosa
  - Tipo U
  - Pseudo-instrução nop
  - Nenhuma ação é realizada



Linguagem de máquina

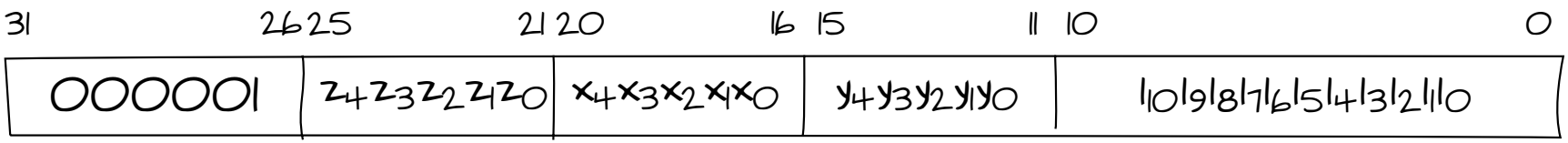
- Operações aritméticas e lógicas
  - Adição (add, addi)
  - Atribuição imediata (mov, movs)
  - Bit a bit (and, or, not, xor)
  - Comparação (cmp, cmpi)
  - Deslocamento (sla, sll, sra, srl)
  - Divisão (div, divs, divi)
  - Multiplicação (mul, muls, muli)
  - Subtração (sub, subi)

Linguagem de máquina

- Operação de atribuição imediata (mov)
  - Tipo U
  - Sem extensão de sinal
  - $R[z] = \overset{11}{0} : x : y : I$

Linguagem de máquina

- Operação de atribuição imediata (movs)
  - Tipo U
  - Com extensão de sinal
  - $R[z] = \overset{11}{x_4} : x : y : l$



Linguagem de máquina

- Operação de adição com registradores (add)
  - Tipo U
  - $R[z] = R[x] + R[y]$

- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$
  - $OV \leftarrow (R[x]_{31} = R[y]_{31} \wedge R[z]_{31} \neq R[x]_{31})$
  - $CY \leftarrow (R[z]_{32} = 1)$

Linguagem de máquina

- Operação de subtração com registradores (sub)
  - Tipo U
  - $R[z] = R[x] - R[y]$

- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$
  - $OV \leftarrow (R[x]_{31} \neq R[y]_{31} \wedge R[z]_{31} \neq R[x]_{31})$
  - $CY \leftarrow (R[z]_{32} = 1)$

Linguagem de máquina

- Operação de multiplicação com registradores (mul)
  - Tipo U
  - Sem sinal
  - $R[I_{4:0}] : R[z] = R[x] \times R[y]$

- Campos afetados
  - $ZN \leftarrow (R[I_{4:0}] : R[z] = 0)$
  - $CY \leftarrow (R[I_{4:0}] \neq 0)$

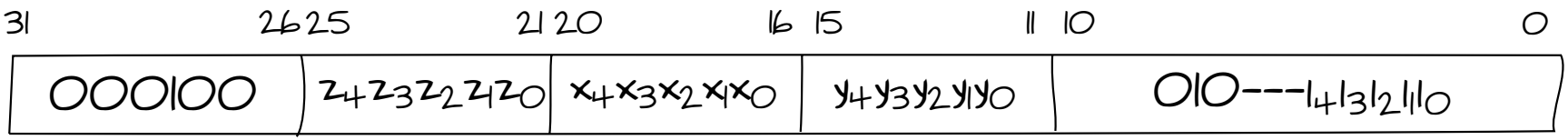
Linguagem de máquina

- Operação de deslocamento para esquerda (sll)
  - Tipo U
  - Lógico (sem sinal)
  - $R[z] : R[x] = (R[z] : R[y]) \times 2^{I_{4:0}+1}$

- Campos afetados
  - $ZN \leftarrow (R[z] : R[x] = 0)$
  - $CY \leftarrow (R[z] \neq 0)$

Linguagem de máquina

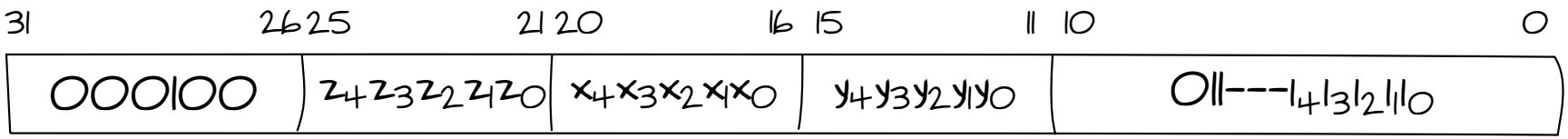
- Operação de multiplicação com registradores (muls)
  - Tipo U
  - Com sinal
  - $R[I_{4:0}] : R[z] = R[x] \times R[y]$



- Campos afetados
  - $ZN \leftarrow (R[I_{4:0}] : R[z] = 0)$
  - $OV \leftarrow (R[I_{4:0}] \neq 0)$

Linguagem de máquina

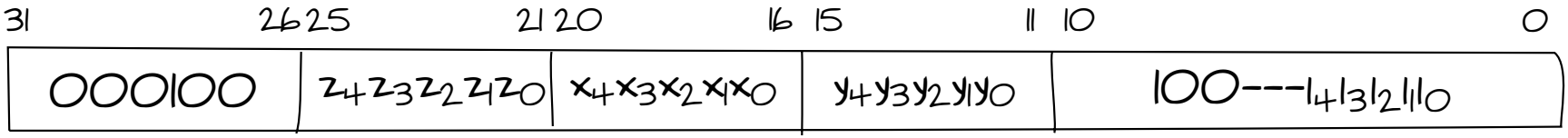
- Operação de deslocamento para esquerda (sla)
  - Tipo U
  - Aritmético (com sinal)
  - $R[z] : R[x] = (R[z] : R[y]) \times 2^{l_{4:0}+1}$



- Campos afetados
  - $ZN \leftarrow (R[z] : R[x] = 0)$
  - $OV \leftarrow (R[z] \neq 0)$

Linguagem de máquina

- Operação de divisão com registradores (div)
  - Tipo U
  - Sem sinal
  - $R[I_{4:0}] = R[x] \bmod R[y]$, $R[z] = R[x] \div R[y]$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $ZD \leftarrow (R[y] = 0)$
  - $CY \leftarrow (R[I_{4:0}] \neq 0)$

Linguagem de máquina

- Operação de deslocamento para direita (srl)
  - Tipo U
  - Lógico (sem sinal)
  - $R[z] : R[x] = (R[z] : R[y]) \div 2^{I_{4:0}+1}$

- Campos afetados
  - $ZN \leftarrow (R[z] : R[x] = 0)$
  - $CY \leftarrow (R[z] \neq 0)$

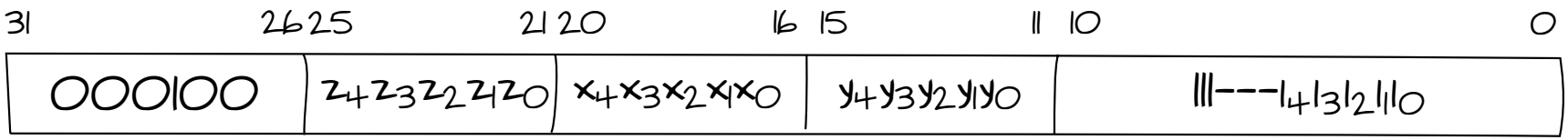
Linguagem de máquina

- Operação de divisão com registradores (divs)
  - Tipo U
  - Com sinal
  - $R[I_{4:0}] = R[x] \bmod R[y]$, $R[z] = R[x] \div R[y]$

- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $ZD \leftarrow (R[y] = 0)$
  - $OV \leftarrow (R[I_{4:0}] \neq 0)$

Linguagem de máquina

- Operação de deslocamento para direita (sra)
  - Tipo U
  - Aritmético (com sinal)
  - $R[z] : R[x] = (R[z] : R[y]) \div 2^{l_{4:0}+1}$



- Campos afetados
  - $ZN \leftarrow (R[z] : R[x] = 0)$
  - $OV \leftarrow (R[z] \neq 0)$

Linguagem de máquina

- Operação de comparação com registradores (cmp)
  - Tipo U
  - $CMP = R[x] - R[y]$

- Campos afetados
  - $ZN \leftarrow (CMP = 0)$
  - $SN \leftarrow (CMP_{31} = 1)$
  - $OV \leftarrow (R[x]_{31} \neq R[y]_{31}) \wedge (CMP_{31} \neq R[x]_{31})$
  - $CY \leftarrow (CMP_{32} = 1)$

Linguagem de máquina

- Operação bit a bit (and)
  - Tipo U
  - $R[z] = R[x]$ **and** $R[y]$

- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$

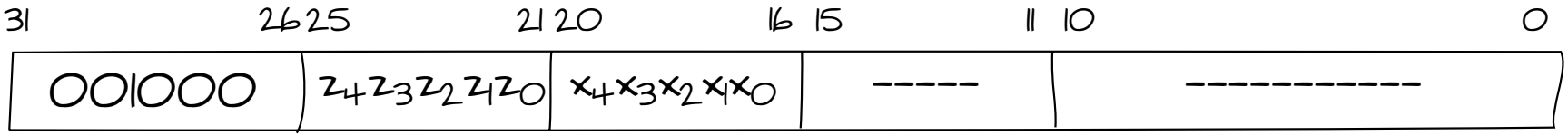
Linguagem de máquina

- Operação bit a bit (or)
  - Tipo U
  - $R[z] = R[x]$ **or** $R[y]$

- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$

Linguagem de máquina

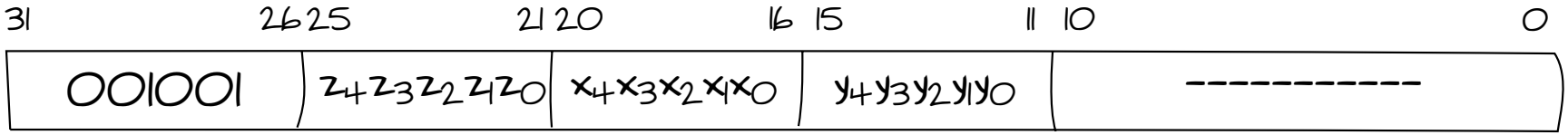
- Operação bit a bit (not)
  - Tipo U
  - $R[z] = \textbf{not}\ R[x]$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$

Linguagem de máquina

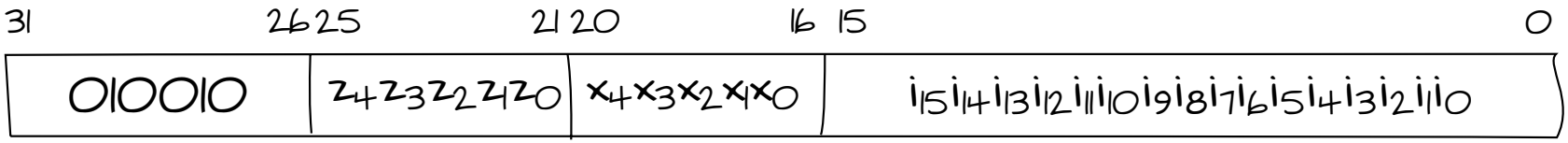
- Operação bit a bit (xor)
  - Tipo U
  - $R[z] = R[x]$ **xor** $R[y]$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$

Linguagem de máquina

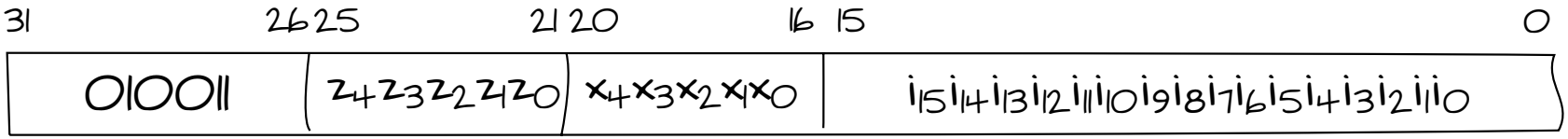
- Operação de adição imediata (addi)
  - Tipo F
  - $R[z] = R[x] + \overset{16}{i_{15}} : i$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$
  - $OV \leftarrow (R[x]_{31} = i_{15}) \wedge (R[z]_{31} \neq R[x]_{31})$
  - $CY \leftarrow (R[z]_{32} = 1)$

Linguagem de máquina

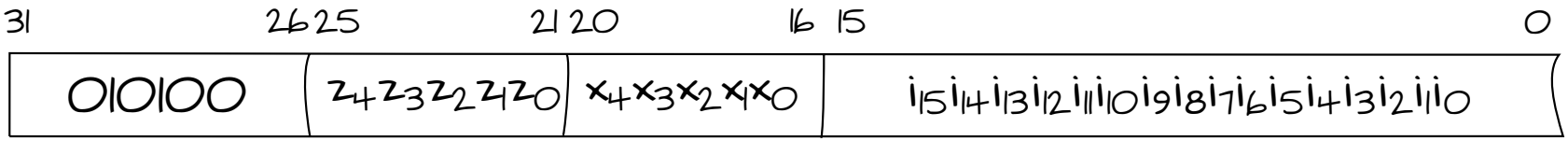
- Operação de subtração imediata (subi)
  - Tipo F
  - $R[z] = R[x] - \overset{16}{i_{15}} : i$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $SN \leftarrow (R[z]_{31} = 1)$
  - $OV \leftarrow (R[x]_{31} \neq i_{15}) \wedge (R[z]_{31} \neq R[x]_{31})$
  - $CY \leftarrow (R[z]_{32} = 1)$

Linguagem de máquina

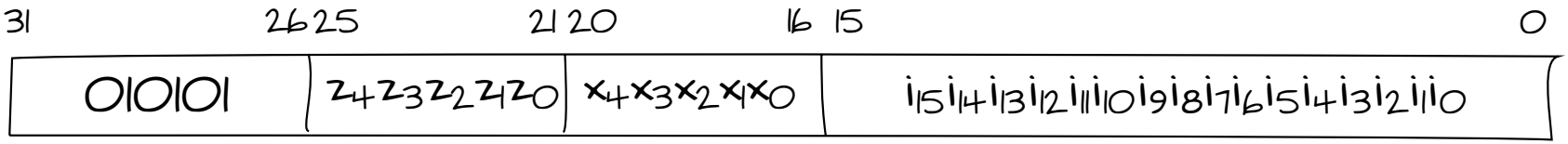
- Operação de multiplicação imediata (muli)
  - Tipo F
  - Com sinal
  - $R[z] = R[x] \times \overset{16}{i_{15}} : i$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $OV \leftarrow (R[z]_{63:32} \neq 0)$

Linguagem de máquina

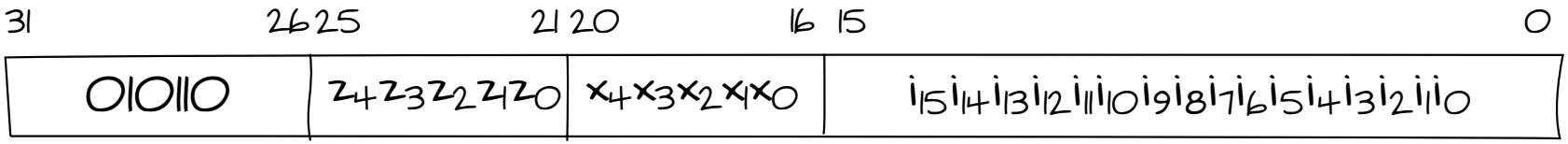
- Operação de divisão imediata (divi)
  - Tipo F
  - Com sinal
  - $R[z] = R[x] \div \overset{16}{i_{15}} : i$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $ZD \leftarrow (i = 0)$
  - $OV \leftarrow 0$

Linguagem de máquina

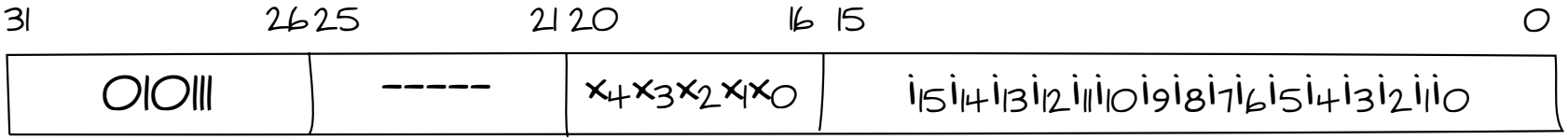
- Operação de resto imediato (modi)
  - Tipo F
  - Com sinal
  - $R[z] = R[x] \bmod i_{15}^{16} : i$



- Campos afetados
  - $ZN \leftarrow (R[z] = 0)$
  - $ZD \leftarrow (i = 0)$
  - $OV \leftarrow 0$

Linguagem de máquina

- Operação de comparação imediata (cmpi)
  - Tipo F
  - $CMPI = R[x] - \overset{16}{i_{15}} : i$



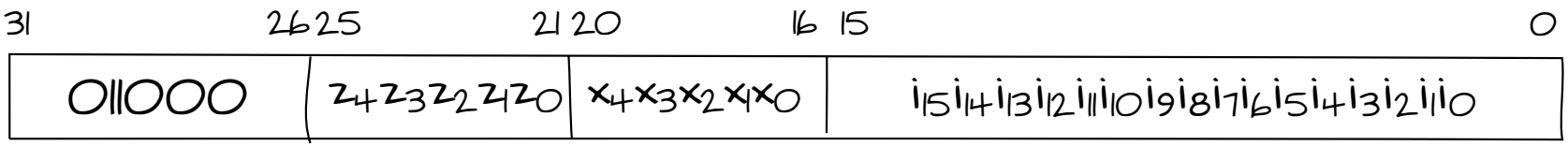
- Campos afetados
  - $ZN \leftarrow (CMPI = 0)$
  - $SN \leftarrow (CMPI_{31} = 1)$
  - $OV \leftarrow (R[x]_{31} \neq i_{15}) \wedge (CMPI_{31} \neq R[x]_{31})$
  - $CY \leftarrow (CMPI_{32} = 1)$

Linguagem de máquina

- Operações de leitura/escrita da memória
  - 8 bits (l8, s8)
  - 16 bits (l16, s16)
  - 32 bits (l32, s32)

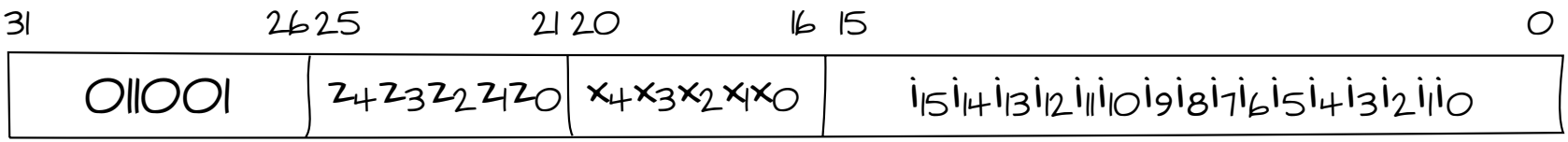
Linguagem de máquina

- Operação de leitura de 8 bits da memória (l8)
  - Tipo F
  - $R[z] = MEM\left[R[x] + i_{15}^{16} : i\right]$



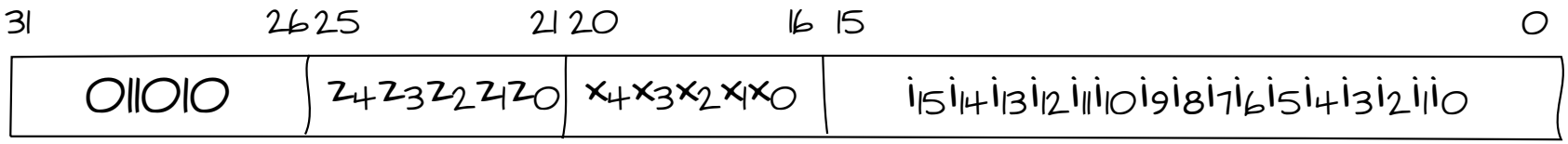
Linguagem de máquina

- Operação de leitura de 16 bits da memória (I16)
  - Tipo F
  - $R[z] = MEM\left[\left(R[x] + i_{15}^{16} : i\right) \ll 1\right]$



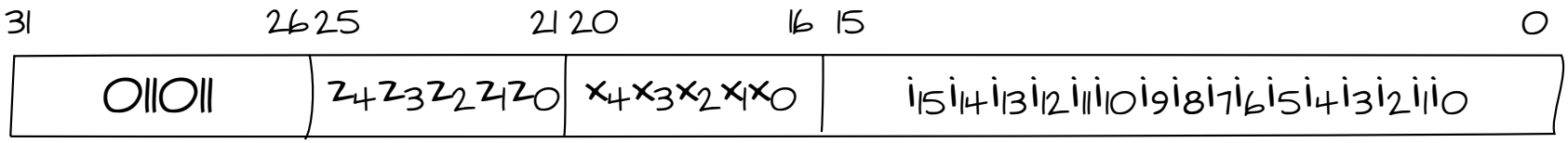
Linguagem de máquina

- Operação de leitura de 32 bits da memória (I32)
  - Tipo F
  - $R[z] = MEM\left[\left(R[x] + i_{15}^{16} : i\right) \ll 2\right]$



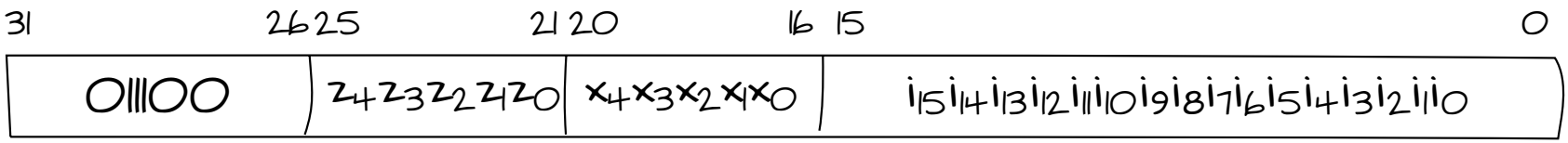
Linguagem de máquina

- Operação de escrita de 8 bits na memória (s8)
  - Tipo F
  - $MEM\left[R[x] + i_{15}^{16} : i\right] = R[z]$



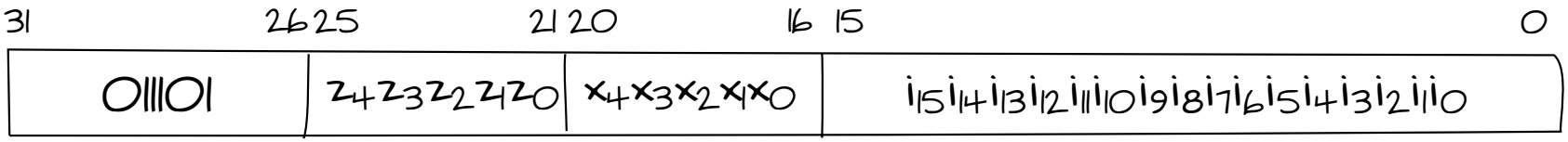
Linguagem de máquina

- Operação de escrita de 16 bits na memória (s16)
  - Tipo F
  - $MEM\left[\left(R[x] + i_{15}^{16} : i\right) \ll 1\right] = R[z]$



Linguagem de máquina

- Operação de escrita de 32 bits na memória (s32)
  - Tipo F
  - $MEM\left[\left(R[x] + i_{15}^{16} : i\right) \ll 2\right] = R[z]$

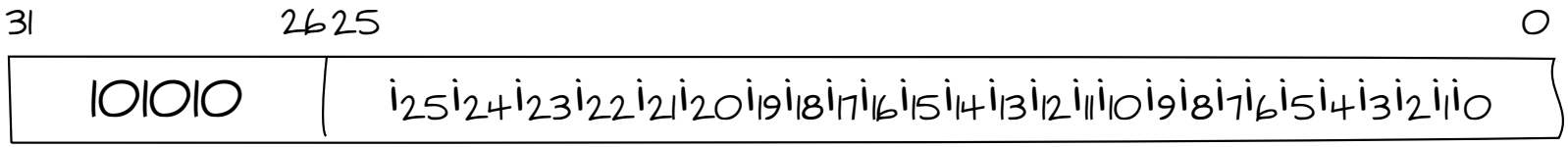


Linguagem de máquina

- Operações de controle de fluxo
  - Desvio condicional (bae, bat, bbe, bbt, beq, bge, bgt, biv, ble, blt, bne, bni, bnz, bzd)
  - Desvio incondicional (bun)
  - Interrupção (int)

Linguagem de máquina

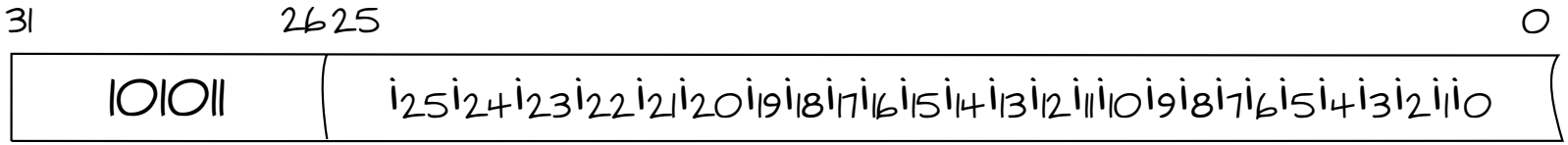
- Operação de desvio condicional (bae)
  - Tipo S
  - Condição *AE* (sem sinal)
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i\right) \ll 2\right]$



$$A \geq B \rightarrow A - B \geq 0 \equiv AE \quad \leftarrow \quad CY = 0$$

Linguagem de máquina

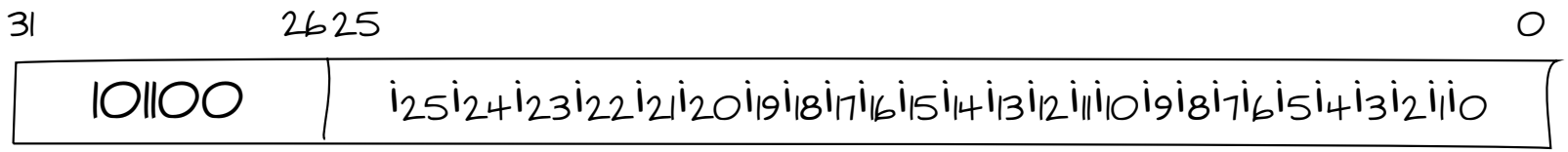
- Operação de desvio condicional (bat)
  - Tipo S
  - Condição *AT* (sem sinal)
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i\right) \ll 2\right]$



$$A > B \rightarrow A - B > 0 \equiv AT \quad \leftarrow \quad (ZN = 0 \wedge CY = 0)$$

Linguagem de máquina

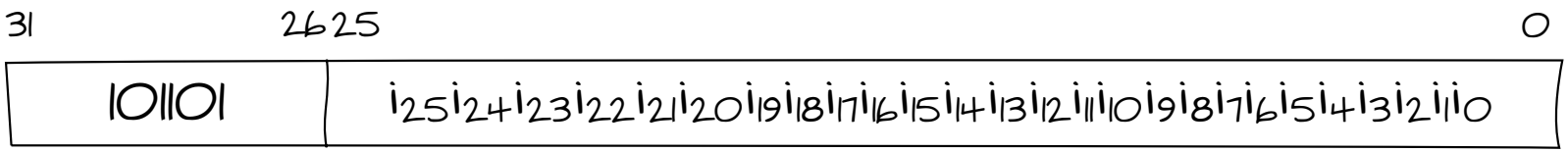
- Operação de desvio condicional (bbe)
  - Tipo S
  - Condição *BE* (sem sinal)
  - $PC = PC + 4 + \left[\left(\overset{6}{i_{25}} : i \right) \ll 2 \right]$



$$A \leq B \rightarrow A - B \leq 0 \equiv BE \quad \leftarrow \quad (ZN = 1 \vee CY = 1)$$

Linguagem de máquina

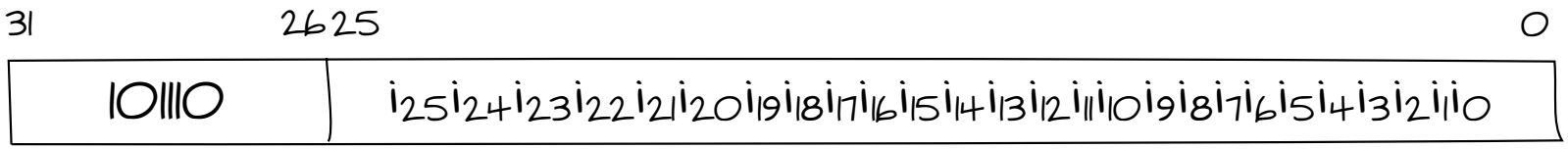
- Operação de desvio condicional (bbt)
  - Tipo S
  - Condição *BT* (sem sinal)
  - $PC = PC + 4 + \left[\left(\overset{6}{i_{25}} : i\right) \ll 2\right]$



$$A < B \rightarrow A - B < 0 \equiv BT \quad \leftarrow \quad CY = 1$$

Linguagem de máquina

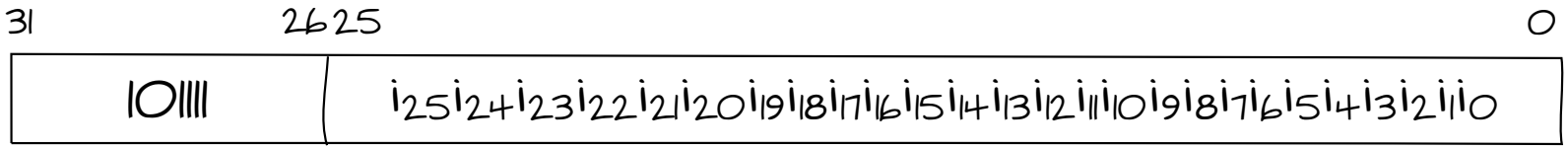
- Operação de desvio condicional (beq)
  - Tipo S
  - Condição *EQ*
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i \right) \ll 2 \right]$



$$A = B \rightarrow A - B = 0 \equiv EQ \quad \leftarrow \quad ZN = 1$$

Linguagem de máquina

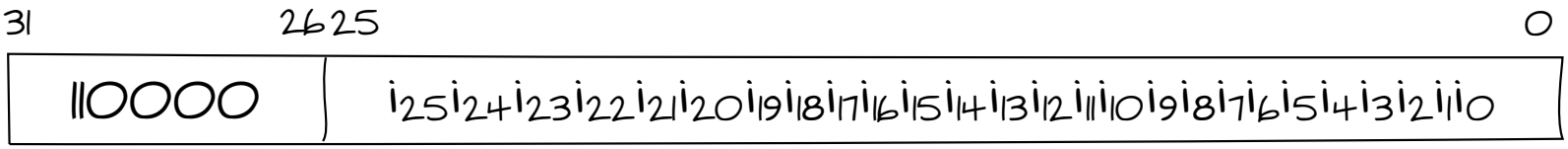
- Operação de desvio condicional (bge)
  - Tipo S
  - Condição *GE* (com sinal)
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i\right) \ll 2\right]$



$$A \geq B \rightarrow A - B \geq 0 \equiv GE \quad \leftarrow \quad SN = OV$$

Linguagem de máquina

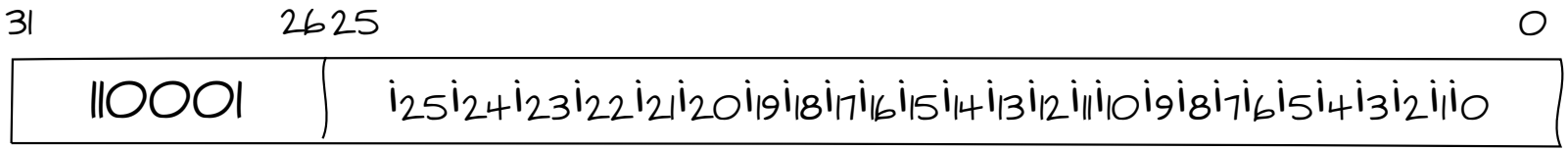
- Operação de desvio condicional (bgt)
  - Tipo S
  - Condição *GT* (com sinal)
  - $PC = PC + 4 + \left[\left({}^{6}_{i_{25}} : i \right) \ll 2 \right]$



$$A > B \rightarrow A - B > 0 \equiv GT \quad \leftarrow \quad (ZN = 0 \wedge SN = OV)$$

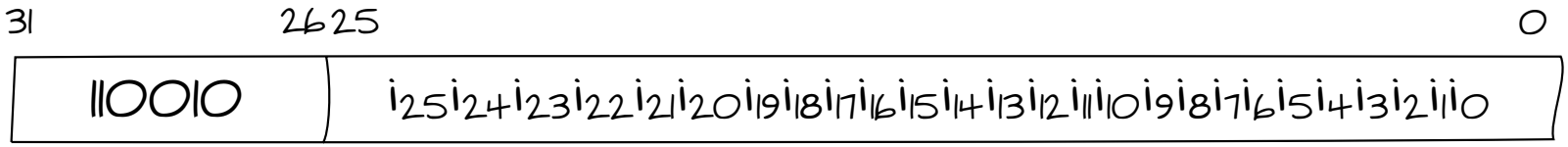
Linguagem de máquina

- Operação de desvio condicional (biv)
  - Tipo S
  - Condição *IV*
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i\right) \ll 2\right]$



Linguagem de máquina

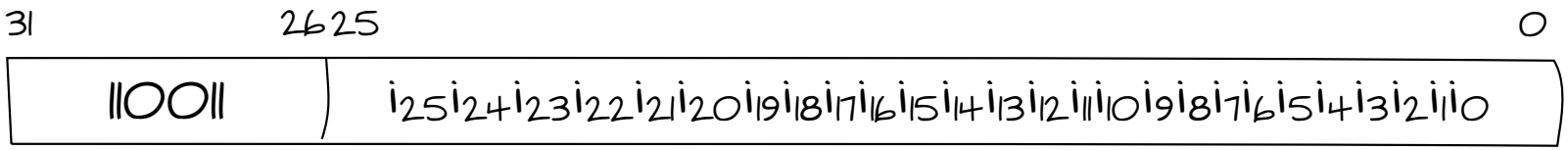
- Operação de desvio condicional (ble)
  - Tipo S
  - Condição *LE* (com sinal)
  - $PC = PC + 4 + \left[\left(\overset{6}{i_{25}} : i\right) \ll 2\right]$



$$A \leq B \rightarrow A - B \leq 0 \equiv LE \quad \leftarrow \quad (ZN = 1 \vee SN \neq OV)$$

Linguagem de máquina

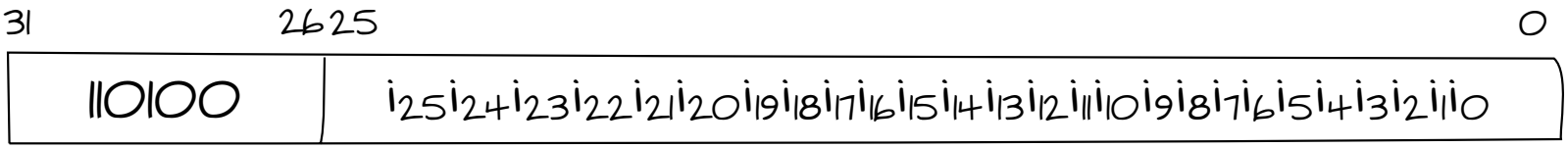
- Operação de desvio condicional (blt)
  - Tipo S
  - Condição *LT* (com sinal)
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i\right) \ll 2\right]$



$$A < B \rightarrow A - B < 0 \equiv LT \quad \leftarrow \quad SN \neq OV$$

Linguagem de máquina

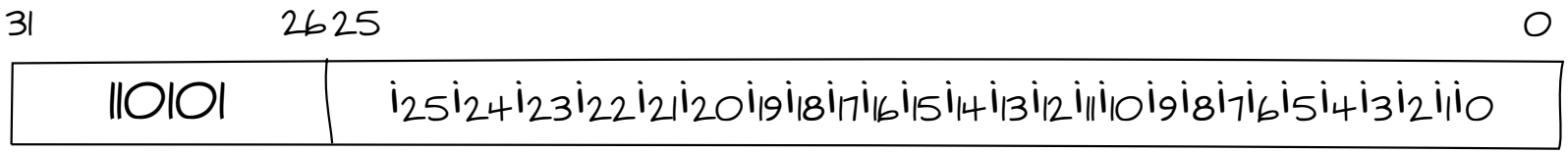
- Operação de desvio condicional (bne)
  - Tipo S
  - Condição *NE*
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i \right) \ll 2 \right]$



$$A \neq B \rightarrow A - B \neq 0 \equiv NE \quad \leftarrow \quad ZN = 0$$

Linguagem de máquina

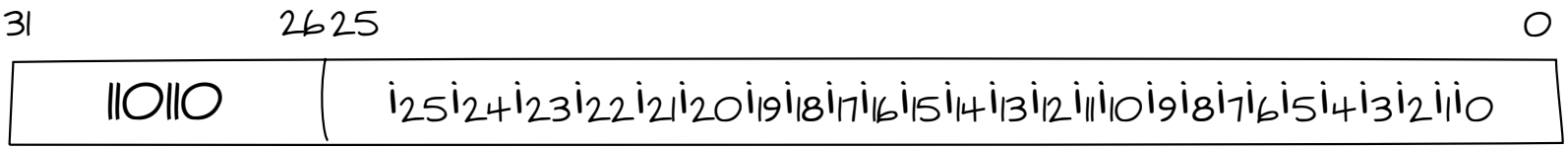
- Operação de desvio condicional (bni)
  - Tipo S
  - Condição *NI*
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i \right) \ll 2 \right]$



$$NI \leftarrow IV = 0$$

Linguagem de máquina

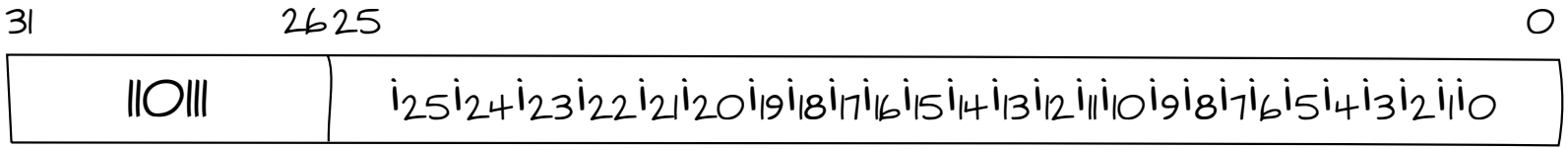
- Operação de desvio condicional (bnz)
  - Tipo S
  - Condição *NZ*
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i \right) \ll 2 \right]$



$$NZ \leftarrow ZD = 0$$

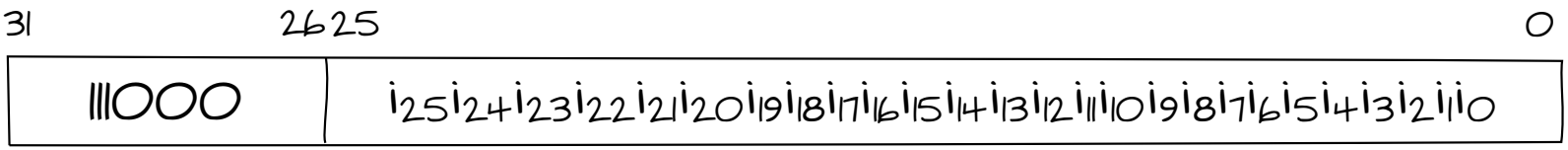
Linguagem de máquina

- Operação de desvio incondicional (bun)
  - Tipo S
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i \right) \ll 2 \right]$



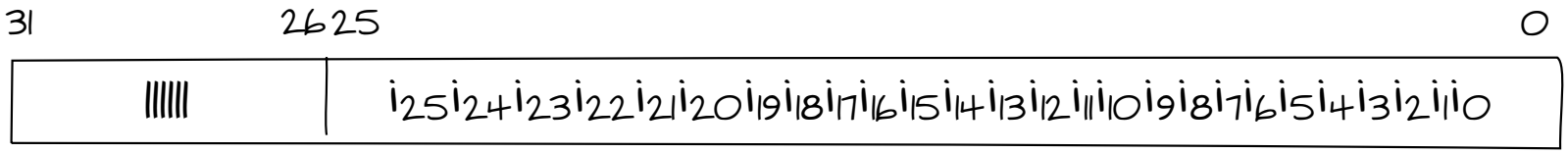
Linguagem de máquina

- Operação de desvio condicional (bzd)
  - Tipo S
  - Condição *ZD*
  - $PC = PC + 4 + \left[\left(i_{25}^{6} : i \right) \ll 2 \right]$



Linguagem de máquina

- Operação de interrupção (int)
  - Tipo S
  - Se $i = 0$, a execução é finalizada



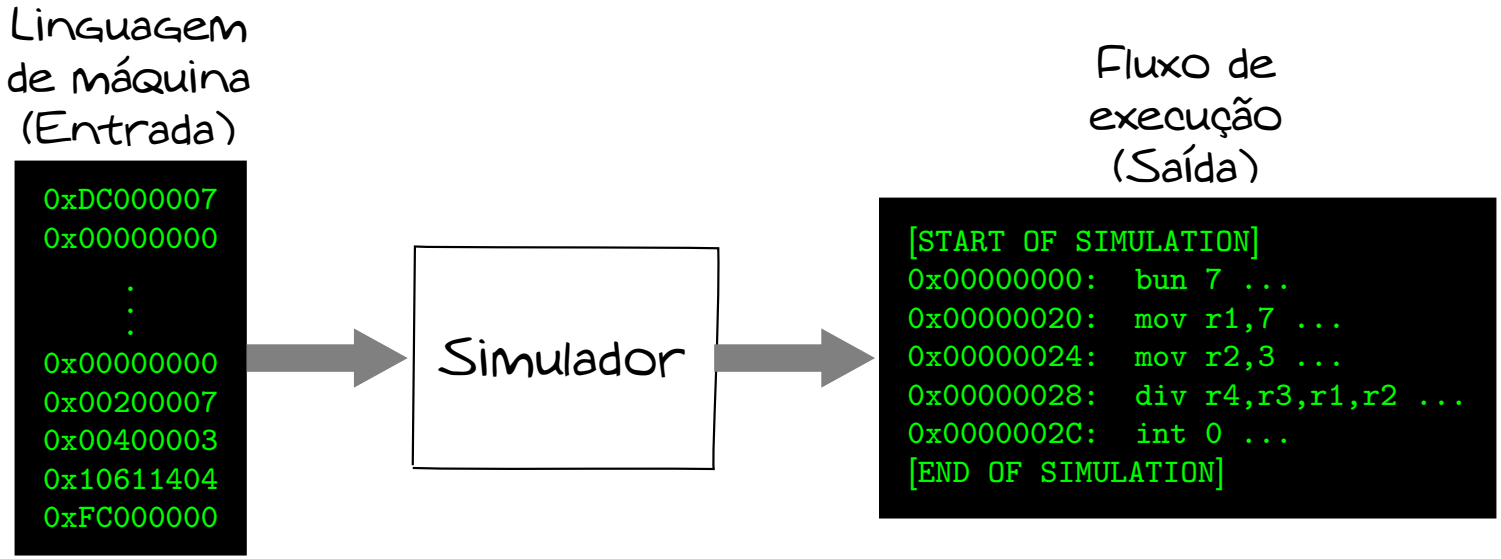
Exemplo

- Considerando o código fonte abaixo, faça sua tradução para código de montagem e depois o converta para linguagem de máquina
  - Execute passo a passo o programa
  - Verifique os resultados obtidos

```
1  // Inteiros com tamanho fixo
2  #include <stdint.h>
3  // Função principal
4  int main() {
5      // Variáveis em memória
6      uint32_t i, n = 5, r = 1;
7      // Controle iterativo
8      for(i = 2; i <= n; i++) {
9          // Multiplicação
10         r = r * i;
11     }
12     // Retorno sem erros
13     return 0;
14 }
```

Exercício

- Considerando a arquitetura Poxim, construa um simulador que realize o carregamento da programação (código binário representado em formato hexadecimal) e execute passo a passo o seu comportamento (fluxo de execução em arquivo)



- Implemente o simulador utilizando as linguagens de programação suportadas, obtendo os argumentos de entrada e de saída por linha de comando