

“No exponential is forever: but ‘Forever’ can be delayed” (Moore; G.; 2003).

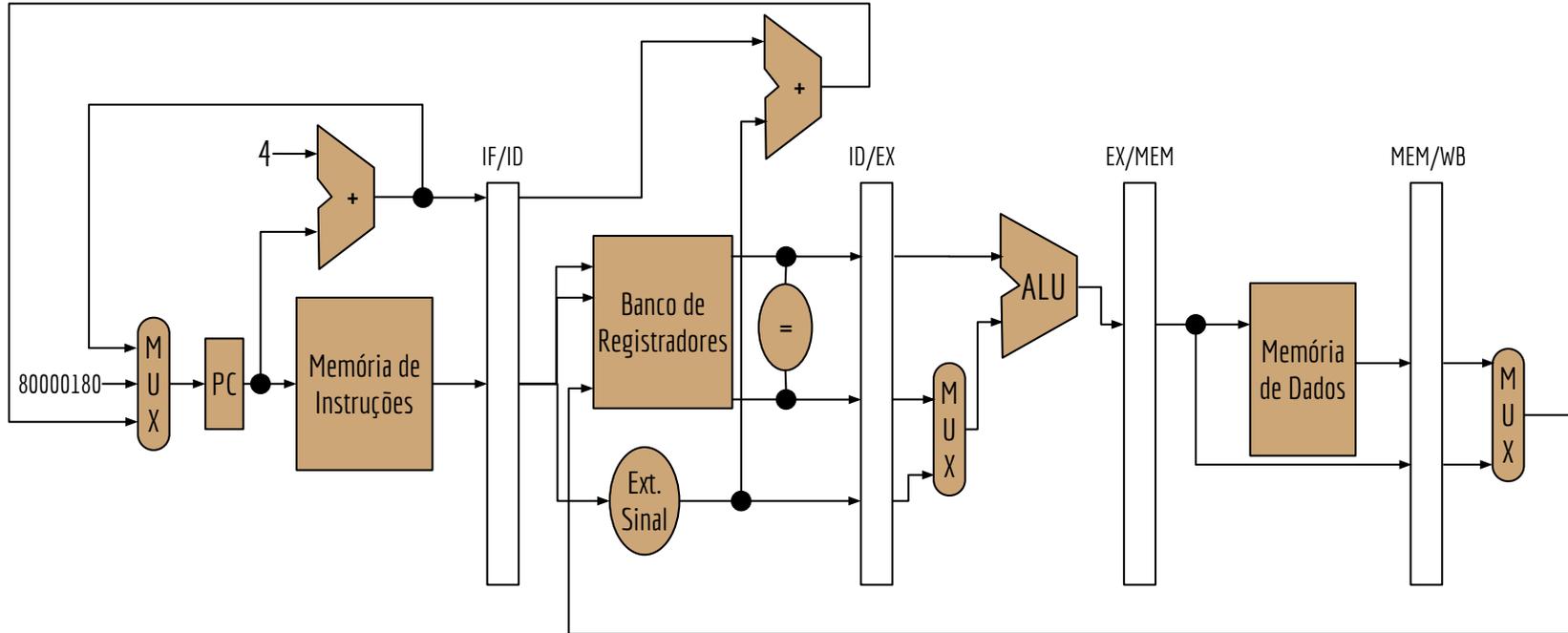
Very Long Instruction Word

Paulo Ricardo Lisboa de Almeida

Pergunta

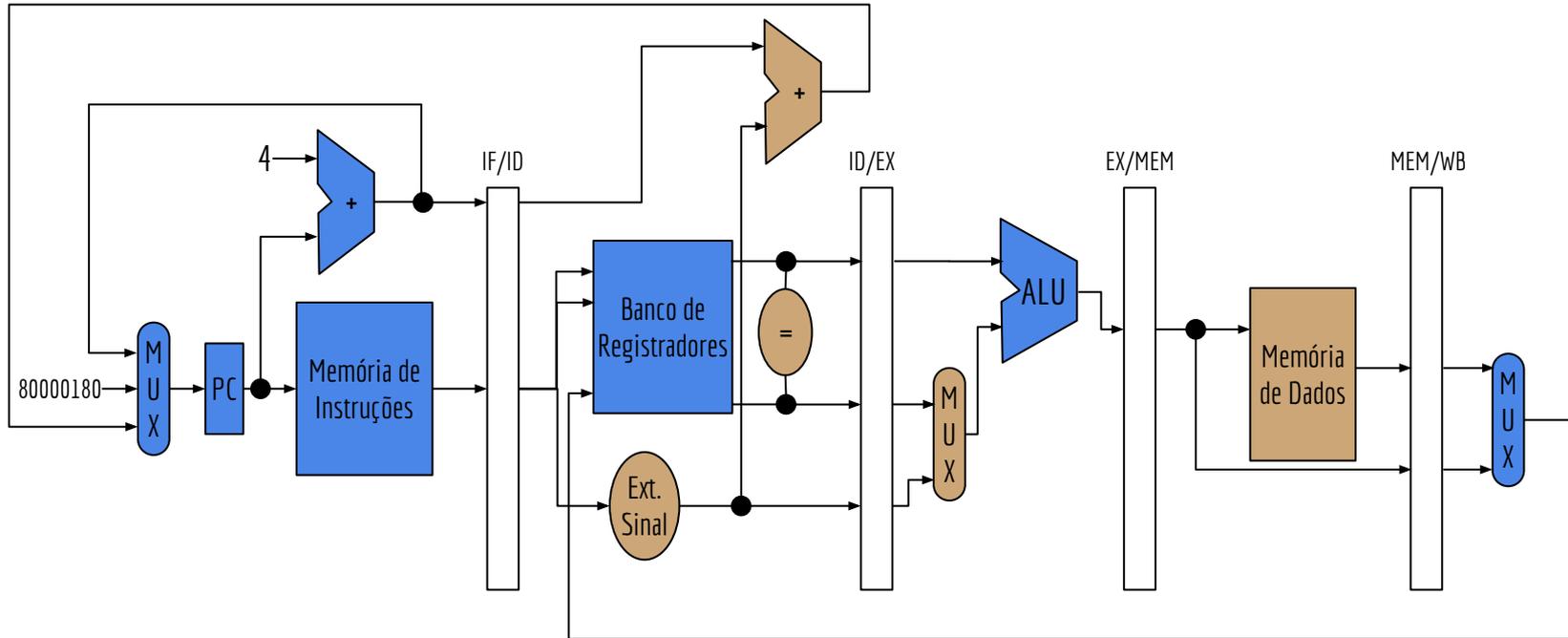
Ao fazer um **add**, quais unidades funcionais são envolvidadas?

Obs.: para simplificar, grande parte dos sinais foi omitida.



Pergunta

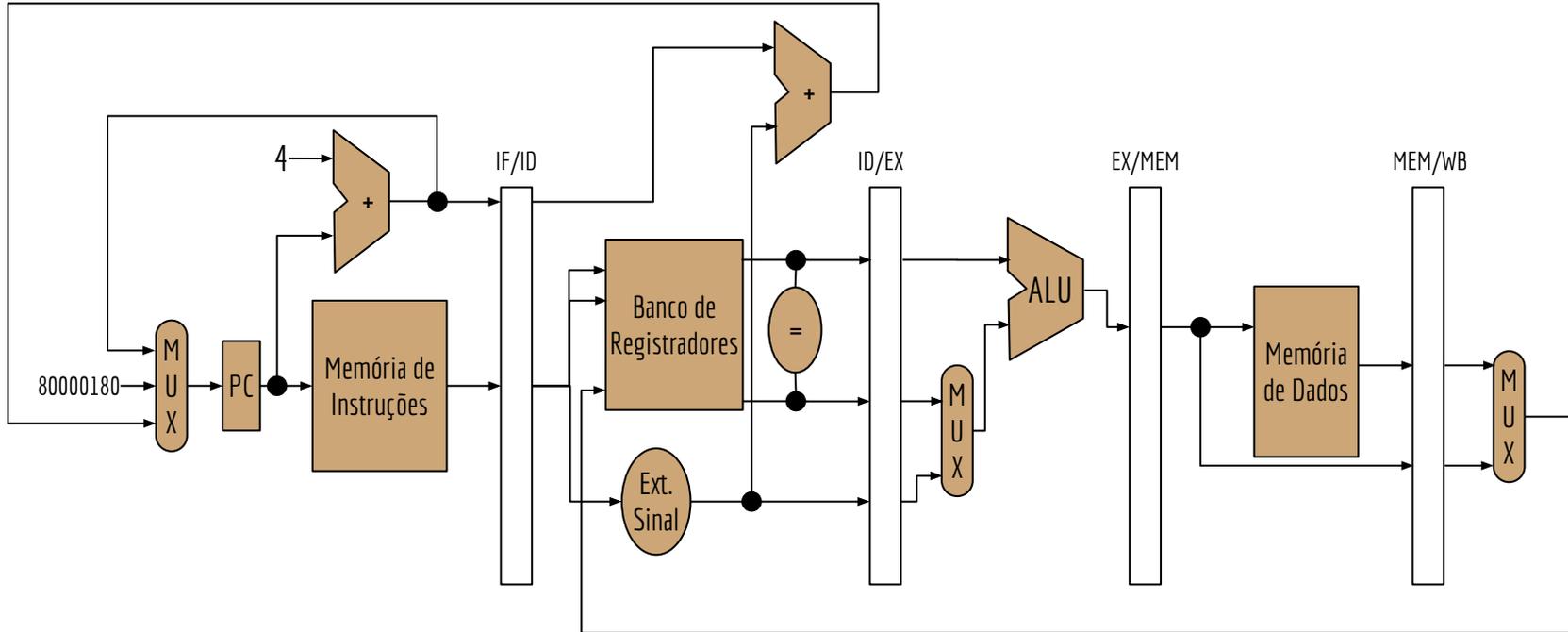
Ao fazer um `add`, quais unidades funcionais são envolvidadas?



Pergunta

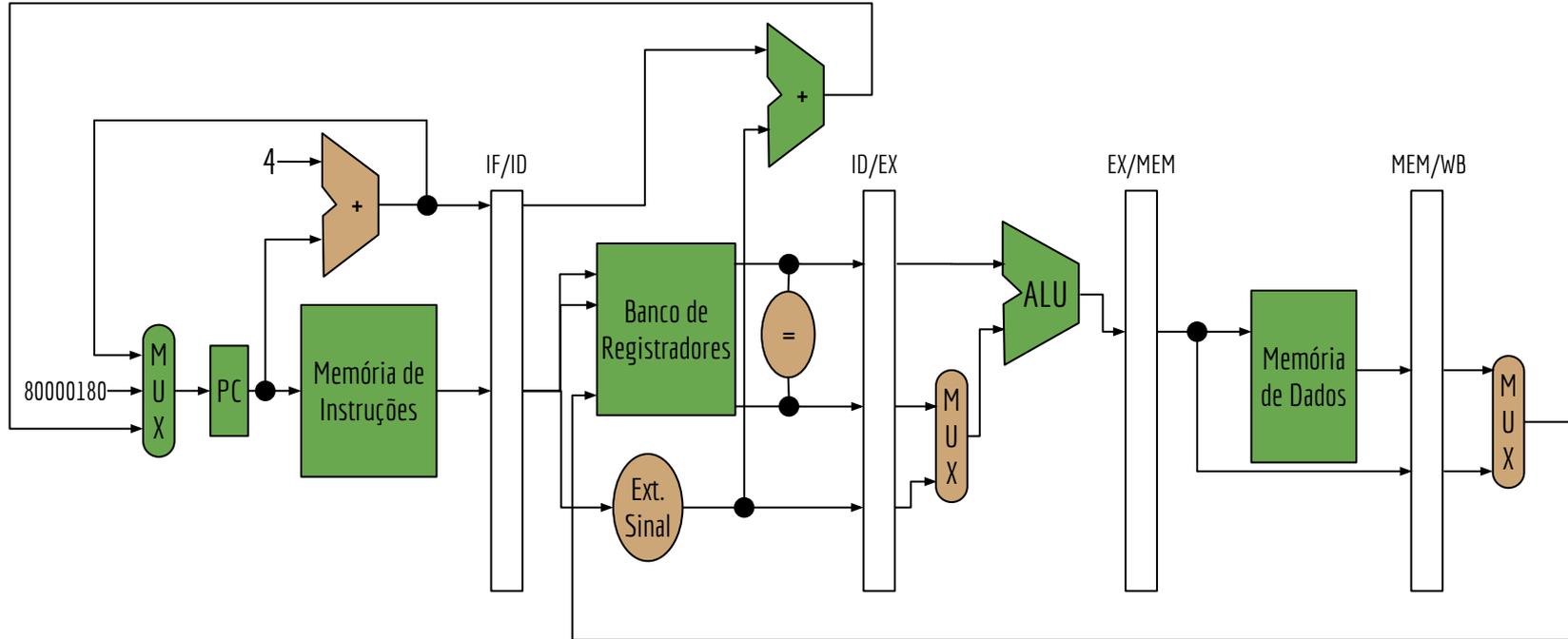
Ao fazer um *sw*, quais unidades funcionais são envolvidadas?

Obs.: para simplificar, boa parte dos sinais foi omitida.



Pergunta

Ao fazer um *sw*, quais unidades funcionais são envolvidadas?



Unidades Ociosas

Dependendo da instrução, algumas unidades funcionais ficam ociosas.

Unidades Ociosas

Dependendo da instrução, algumas unidades funcionais ficam ociosas.

Uma forma de sanar isso é enviar duas instruções simultaneamente em um único “pacote”.

O “pacote” contém duas (ou +) instruções, que deve ser uma combinação específica de tipos de instrução.

VLIW

VLIW - Very Long Instruction Word.

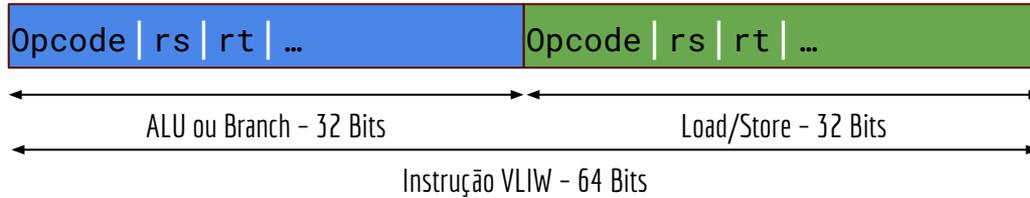
Combinar duas ou mais instruções em uma única instrução longa.

VLIW

VLIW - Very Long Instruction Word.

Combinar duas ou mais instruções em uma única instrução longa.

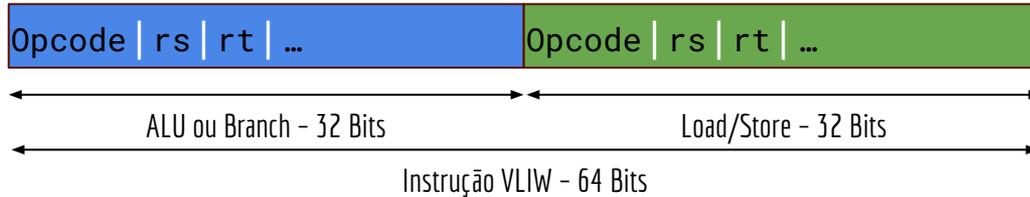
Exemplo: pacotes contendo uma combinação de instruções de aritméticas ou branch + um load/store.



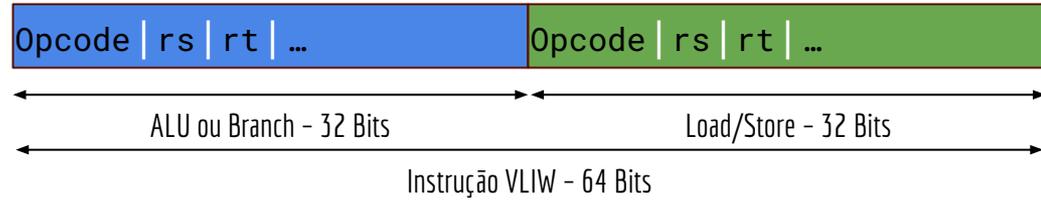
VLIW

No modelo VLIW, o **compilador/programador** se encarrega de montar as instruções corretamente nos pacotes.

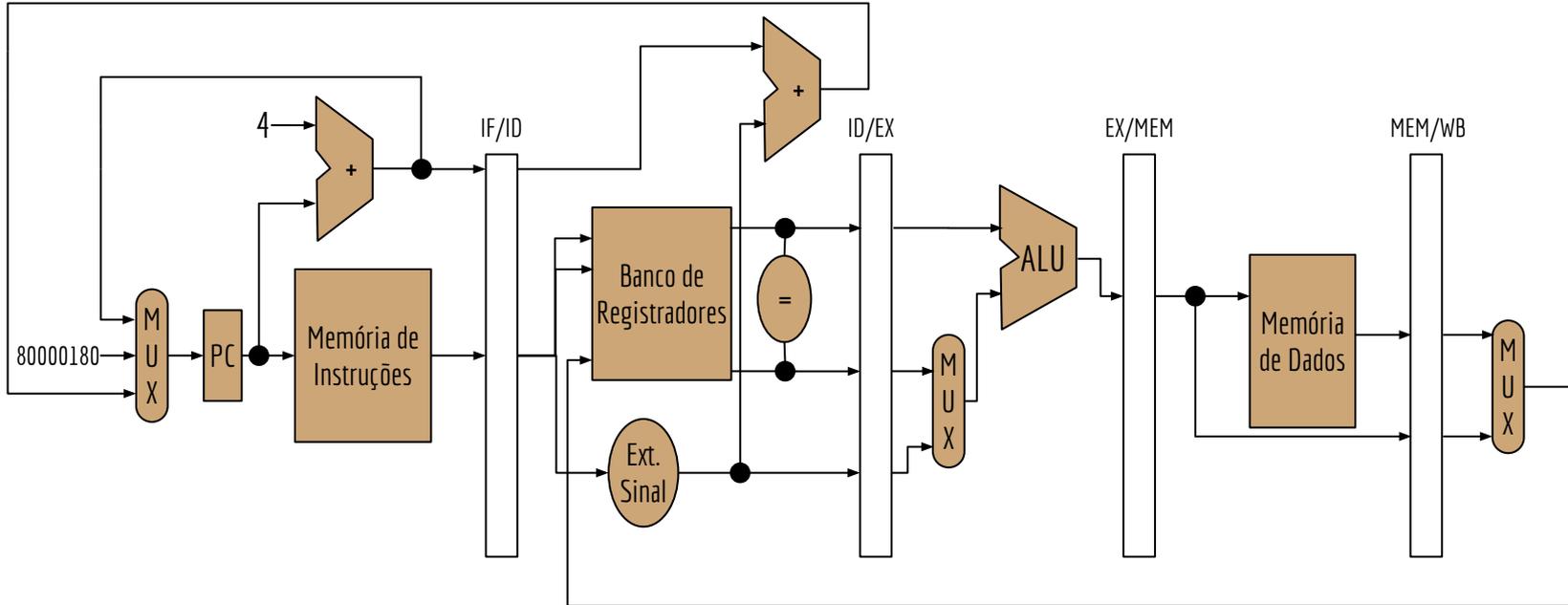
Se não for possível montar a combinação de instruções, uma delas pode ser substituída por um **nop**.



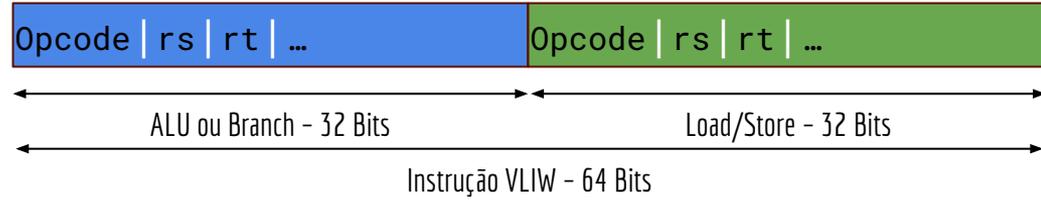
Instruction Fetch



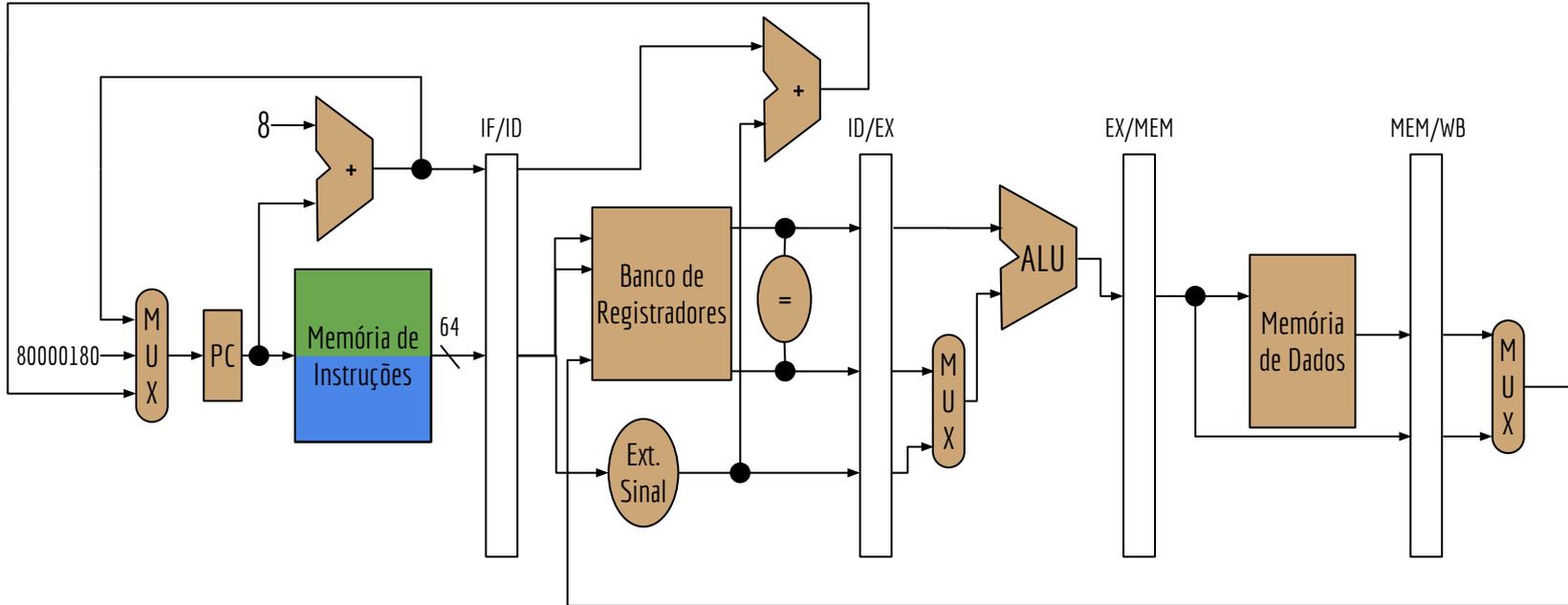
Começando pelo estágio *Instruction Fetch*, o que precisamos modificar?



Instruction Fetch

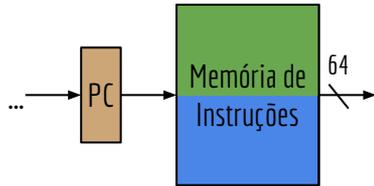


No IF, a memória de instruções precisa servir 64 bits.

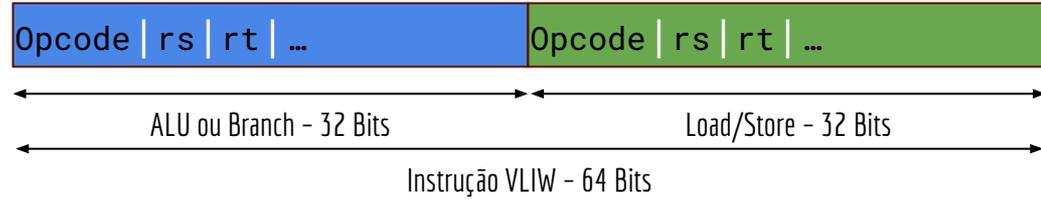


Instruções de 64 bits

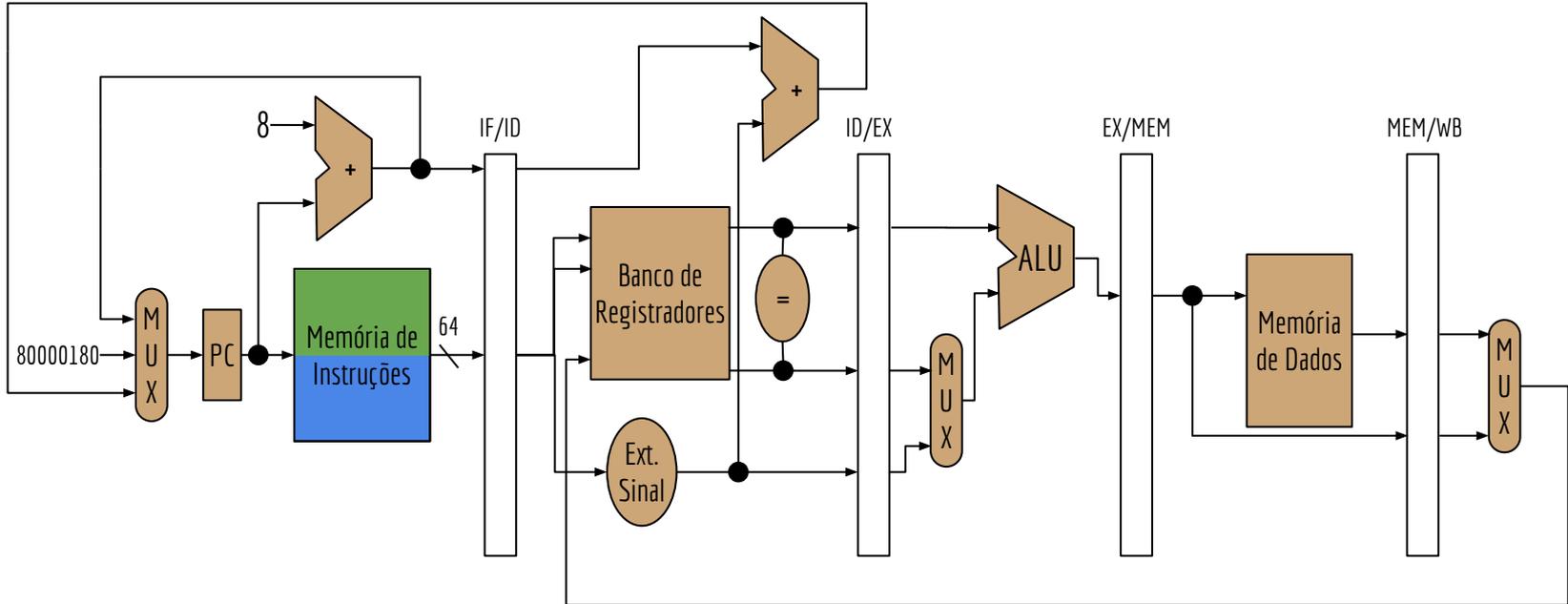
Para facilitar, o que geralmente se faz é alinhar as instruções em endereços múltiplos de n bits na memória.
 n é o tamanho da instrução (no caso, 64 bits).



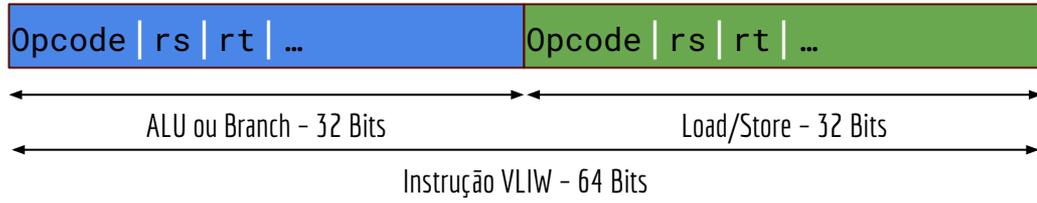
Instruction Decode



E no estágio *Instruction Decode*?



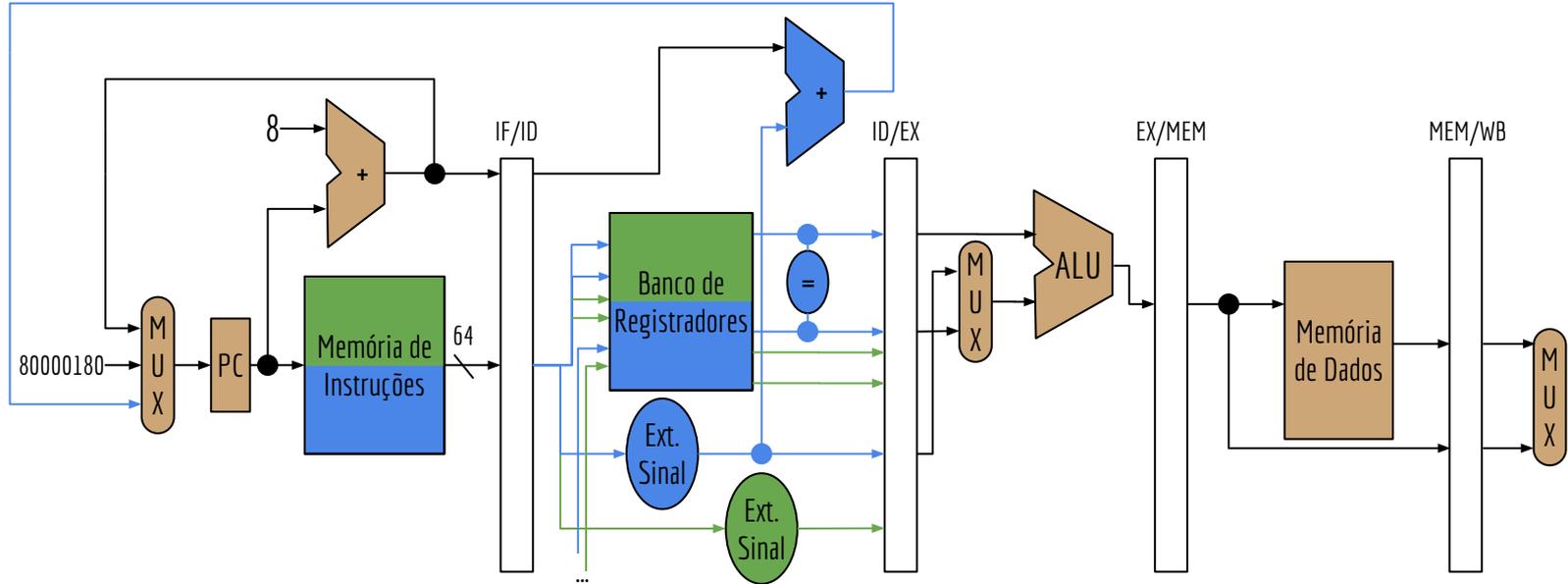
Instruction Decode



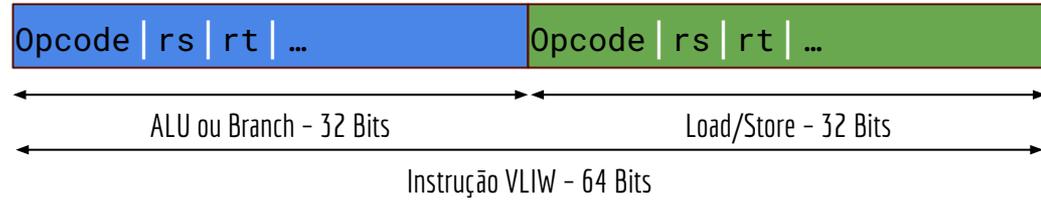
Banco de registradores precisa ler 4 registradores (e.g., add + lw).

Banco de registradores precisa escrever 2 registradores (e.g., add + sw).

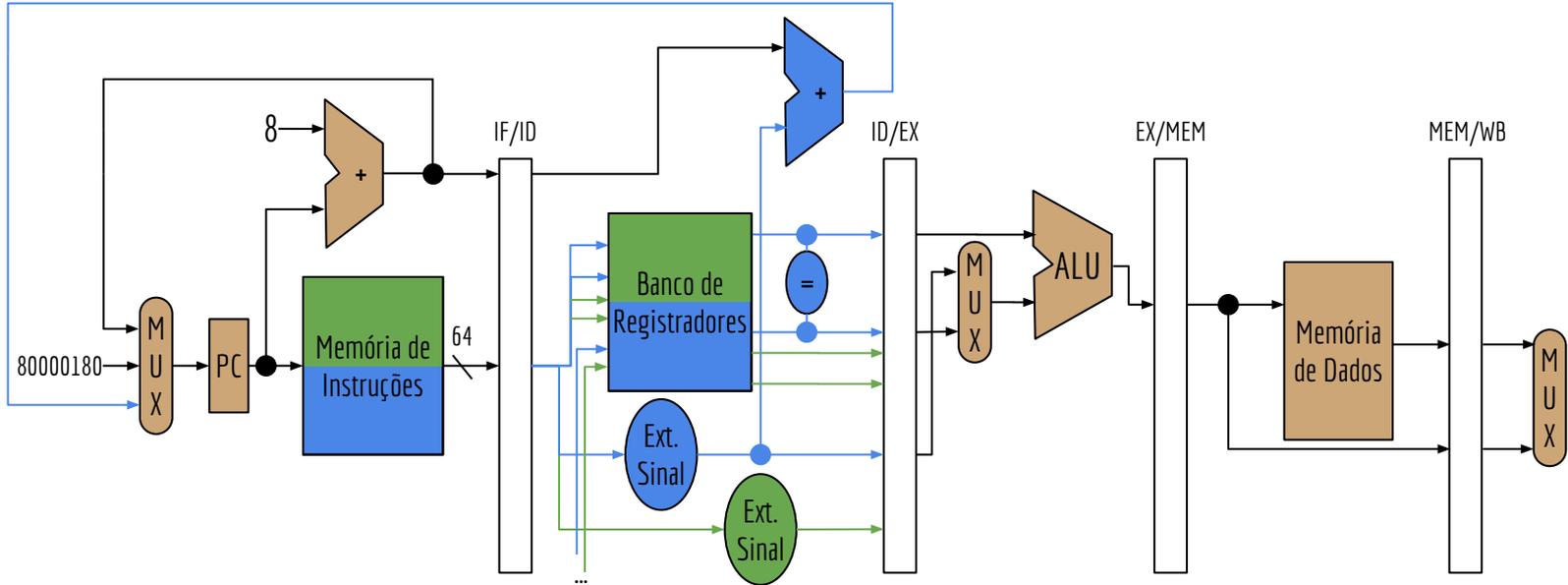
Unidade de extensão de sinal extra (e.g., addi + lw).



Execution

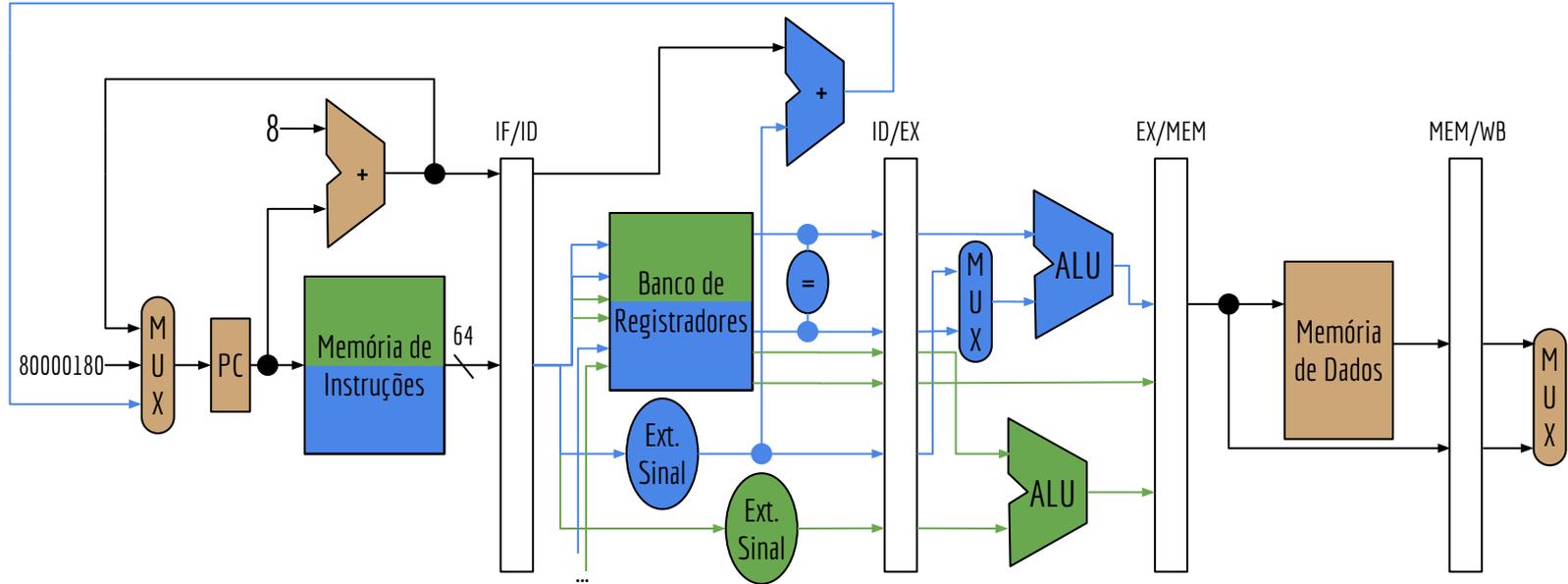
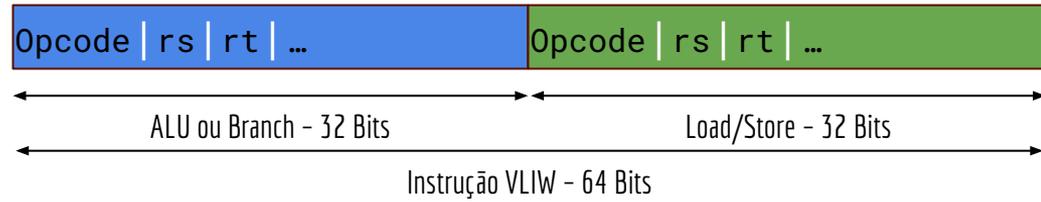


E no estágio *Execution*?



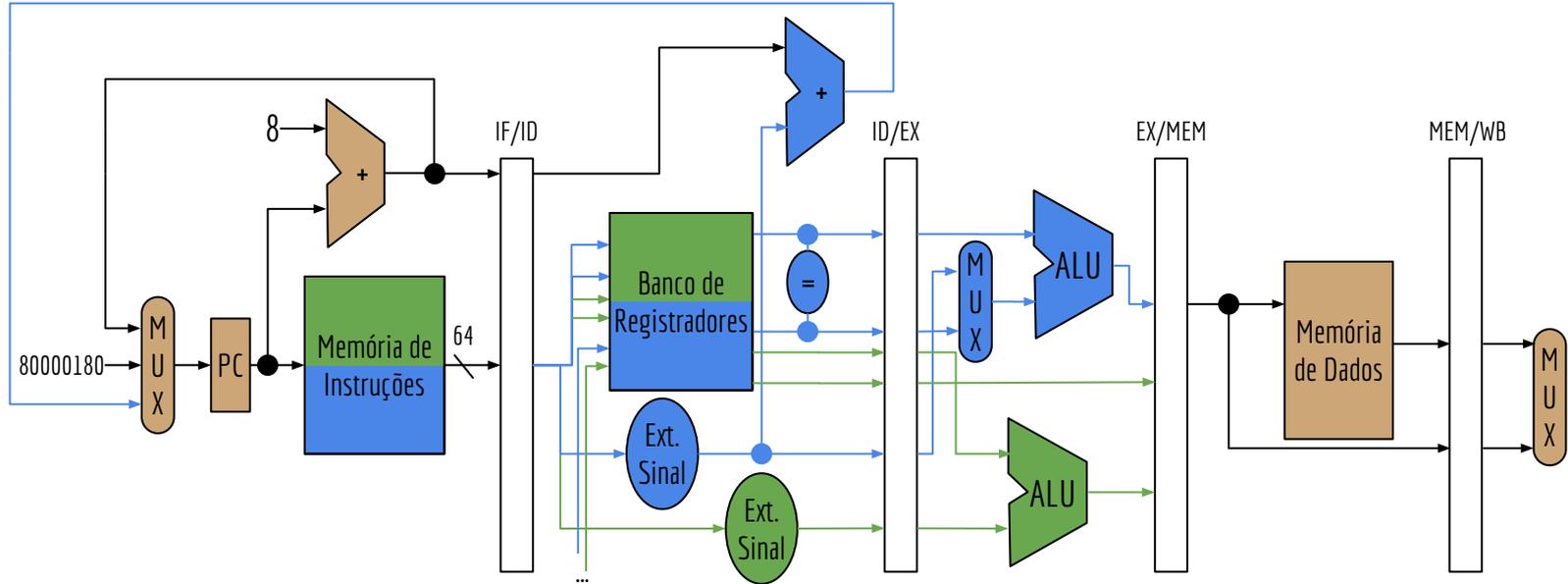
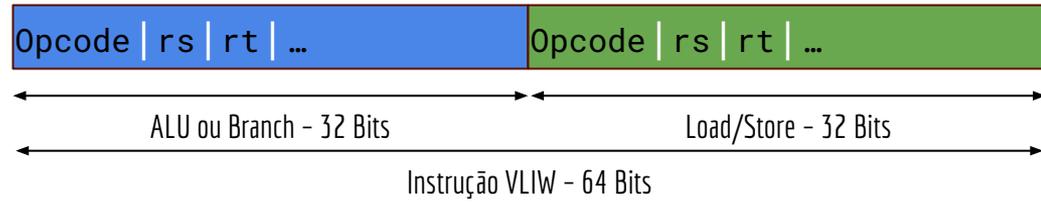
Execution

Duas ALUs. Uma para a primeira instrução (e.g., add), e uma para calcular o endereço efetivo do load/store.



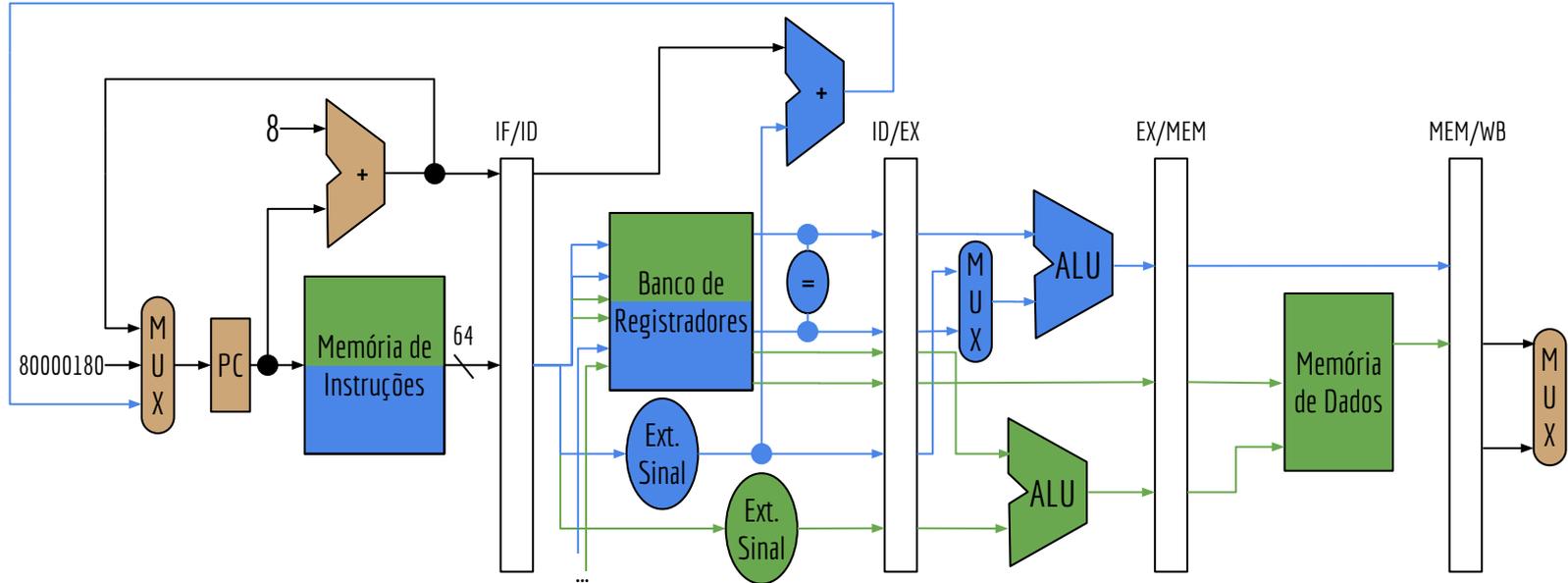
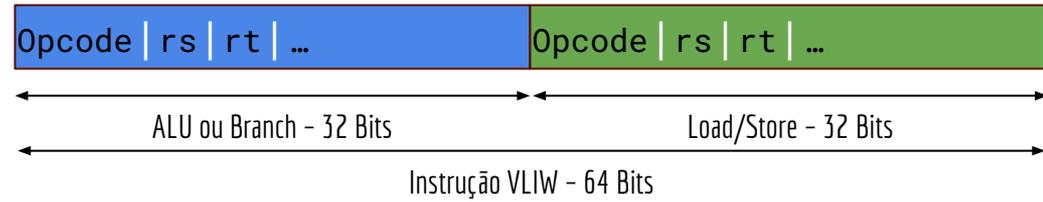
Memory

E no estágio *Memory*?



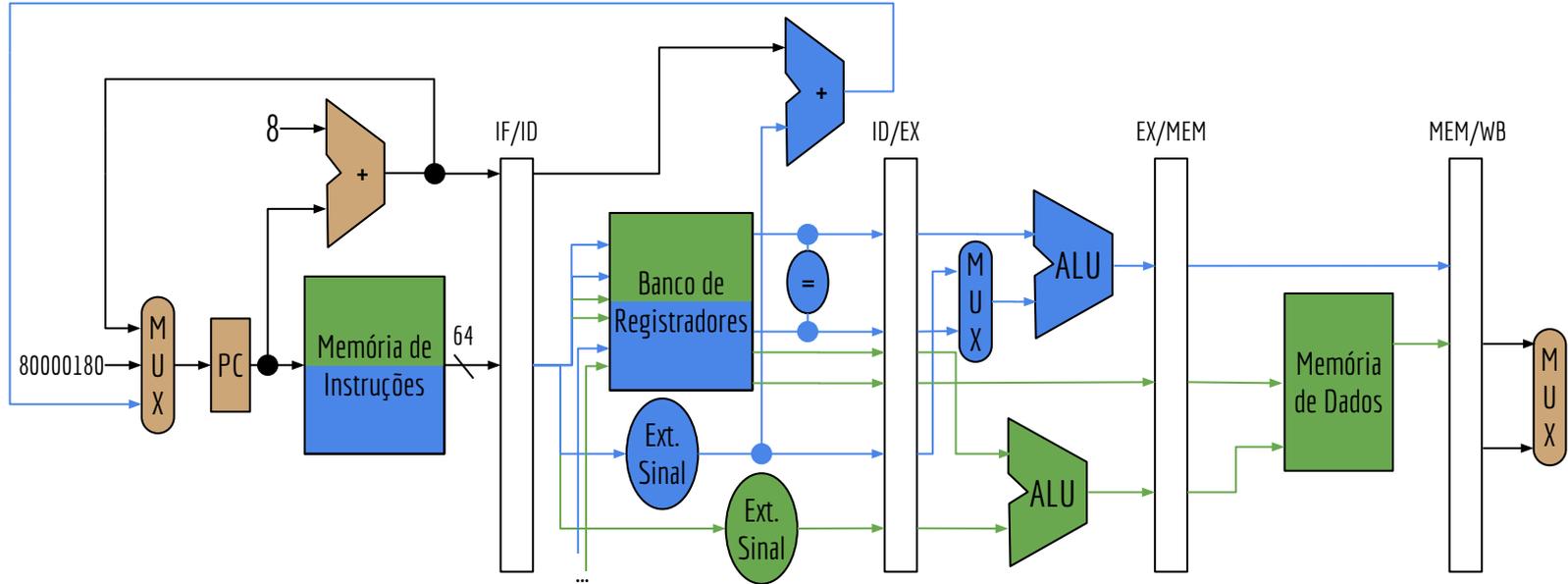
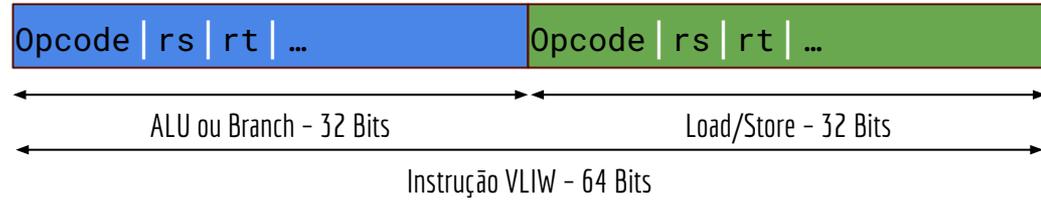
Memory

Apenas algumas correções nos barramentos é o suficiente.



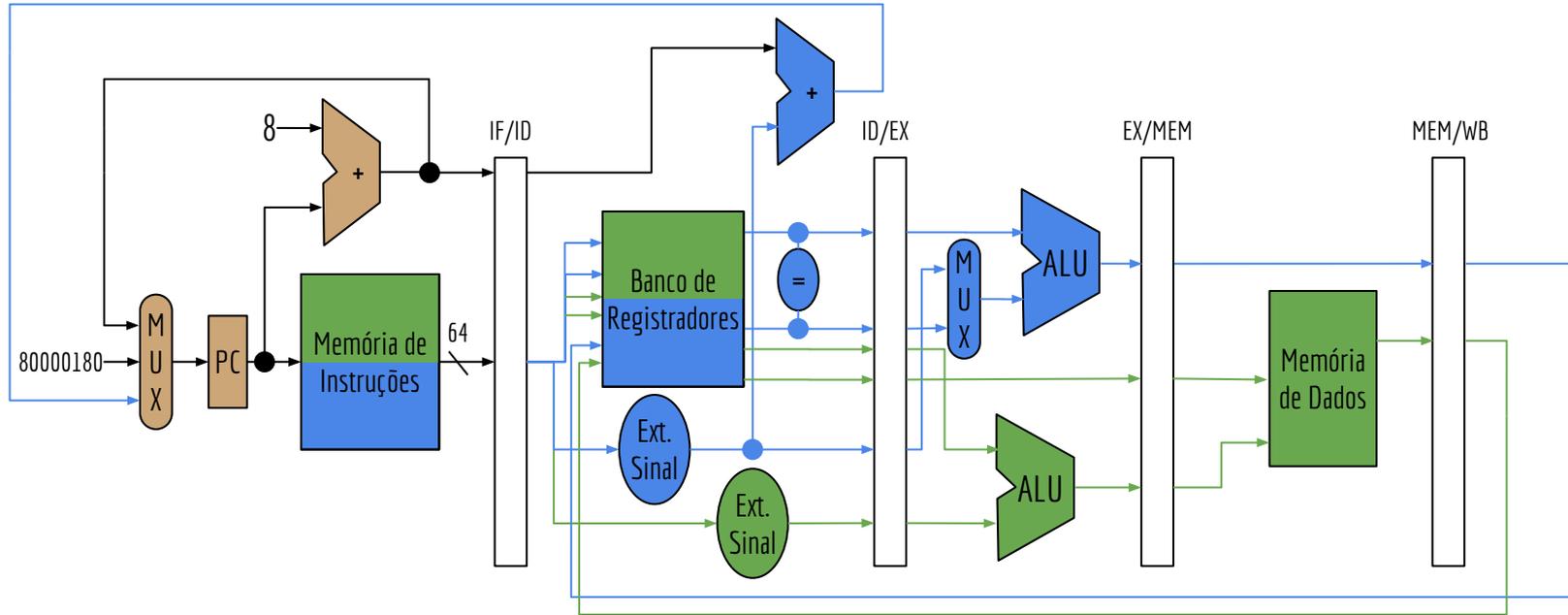
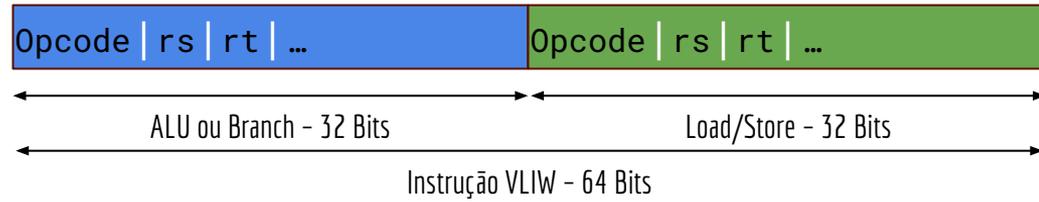
Write Back

E no estágio *Write Back*?



Write Back

A principal mudança era a inclusão de duas portas de escrita no banco de registradores (já feito). Resta somente mudar alguns barramentos.



VLIW

Com apenas algumas mudanças e adição de alguns componentes, a CPU é agora capaz de processar um “pacote” contendo duas instruções em paralelo, sem gerar hazards estruturais.

Qual o CPI da CPU considerando que não geramos stalls?

VLIW

Com apenas algumas mudanças e adição de alguns componentes, a CPU é agora capaz de processar um “pacote” contendo duas instruções em paralelo, sem gerar hazards estruturais.

Qual o CPI da CPU considerando que não geramos stalls?

0,5.

Conseguimos manter a CPU sem stalls? Qual a dificuldade em manter o pipeline cheio?

VLIW

Com apenas algumas mudanças e adição de alguns componentes, a CPU é agora capaz de processar um “pacote” contendo duas instruções em paralelo, sem gerar hazards estruturais.

Qual o CPI da CPU considerando que não geramos stalls?

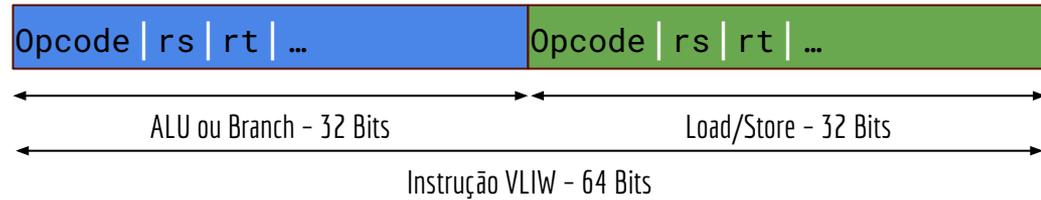
0,5.

Conseguimos manter a CPU sem stalls? Qual a dificuldade em manter o pipeline cheio?

O compilador vai precisar encontrar combinações de instruções para despachar em pares.

- Sempre que não for possível, tempo é jogado fora com um **nop** em um dos “lados do pacote”.
- A compilação/montagem é mais complicada.
- O controle é mais complicado, e temos duas instruções que podem gerar hazards em paralelo no pipeline.

Faça você mesmo



Considere um processador VLIW que aceita as instruções no formato exemplificado durante a aula. Reorganize as instruções a seguir para que elas sejam despachadas em pacotes. Você pode usar **nops**.

```
ori $s2,$zero,160 #numero elementos * 4 bytes
lw $s1,0($sp) #carrega a base de $s1 da pilha
add $s2,$s1, $s2 #final do vetor
loop: lw $t0,0($s1)
      addu $t0,$t0,$s2
      sw $t0,0($s1)
      addi $s1,$s1,4
      bne $s1,$s2,loop
```

Faça você mesmo



loop:

```
ori $s2,$zero,160  
add $s2,$s1,$s2  
nop  
addu $t0,$t0,$s2  
addi $s1,$s1,4  
bne $s1,$s2,Loop
```

```
lw $s1,0($sp)  
nop  
lw $t0,0($s1)  
nop  
nop  
sw $t0,-4($s1)
```

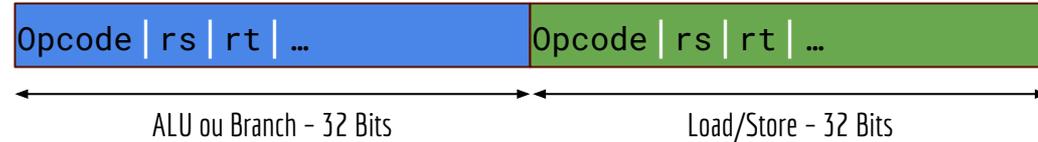
Faça você mesmo

Considere o seguinte trecho de um programa em C. Considere que o vetor já está alocado e inicia no endereço 0x10000000. Faça uma versão do trecho em Assembly do MIPS32, e uma versão do Assembly VLIW do MIPS32 de exemplo dado em aula.

```
int vetor[10];  
for(int i = 0; i < 10; i++)  
    vetor[i] = vetor[i] + 1;  
//...
```

Faça você mesmo

```
int vetor[10];
for(int i = 0; i < 10; i++)
    vetor[i] = vetor[i]+1;
//...
```



```
ori $s0,$zero,0 #inicio do vetor
lui $s0,0x1000
ori $s1,$zero,40 #contador regressivo
```

loop:

```
addi $s1,$s1,-4 #decrementa cont.
addi $s2,$s0,$s1 #endereço efetivo
lw $t0,0($s2)
addi $t0,$t0,1
sw $t0,0($s2)
bne $s1,$zero, loop
```

```
ori $s0,$zero,0
lui $s0,0x1000
ori $s1,$zero,40
```

loop:

```
addi $s2,$s0,$s1
nop
addi $s1,$s1,-8
addi $t0,$t0,1
addi $t1,$t1,1
bne $s1,$zero, loop
```

```
nop
nop
nop
```

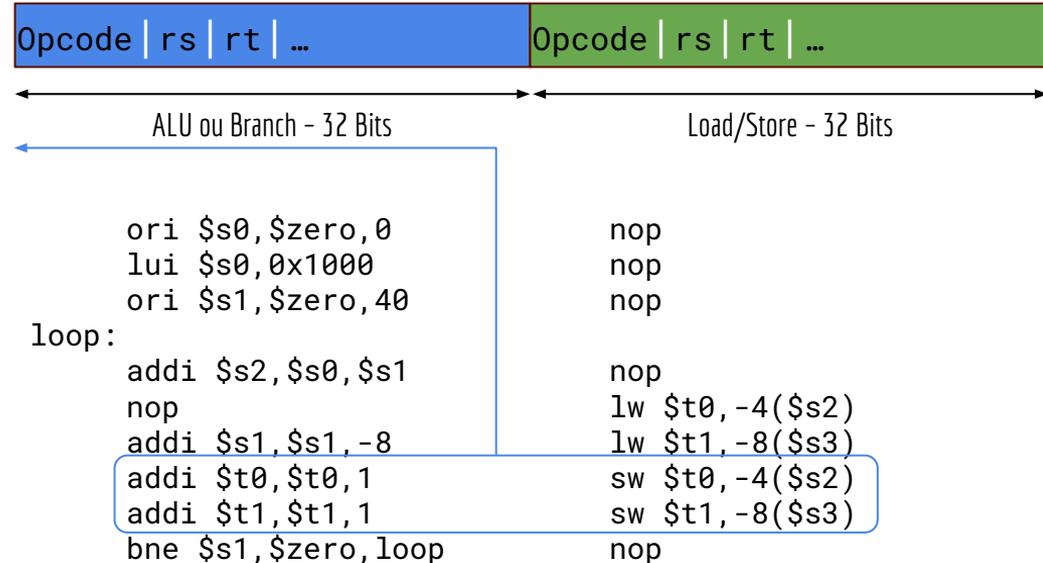
```
nop
lw $t0,-4($s2)
lw $t1,-8($s3)
sw $t0,-4($s2)
sw $t1,-8($s3)
nop
```

Faça você mesmo

```
int vetor[10];  
for(int i = 0; i < 10; i++)  
    vetor[i] = vetor[i]+1;  
//...
```

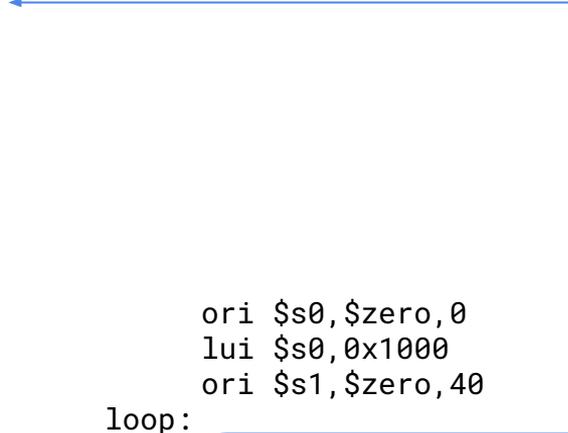
Assumindo que temos unidades de forwarding capazes de redirecionar o resultado de \$t0/\$t1 para o sw.

```
ori $s0,$zero,0 #inicio do vetor  
lui $s0,0x1000  
ori $s1,$zero,40 #contador regressivo  
loop:  
addi $s1,$s1,-4 #decrementa cont.  
addi $s2,$s0,$s1 #endereço efetivo  
lw $t0,0($s2)  
addi $t0,$t0,1  
sw $t0,0($s2)  
bne $s1,$zero, loop
```



Faça você mesmo

Mesmo número de instruções dentro do loop, mas cada iteração faz o dobro de trabalho.



```
ori $s0,$zero,0 #inicio do vetor
lui $s0,0x1000
ori $s1,$zero,40 #contador regressivo
```

loop:

```
addi $s1,$s1,-4 #decrementa cont.
addi $s2,$s0,$s1 #endereço efetivo
lw $t0,0($s2)
addi $t0,$t0,1
sw $t0,0($s2)
bne $s1,$zero, loop
```

```
ori $s0,$zero,0
lui $s0,0x1000
ori $s1,$zero,40
```

loop:

```
addi $s2,$s0,$s1
nop
addi $s1,$s1,-8
addi $t0,$t0,1
addi $t1,$t1,1
bne $s1,$zero, loop
```

```
nop
nop
nop
```

```
nop
lw $t0,-4($s2)
lw $t1,-8($s3)
sw $t0,-4($s2)
sw $t1,-8($s3)
nop
```

Loop Unrolling

Fizemos um **loop unrolling**.

Pesquise sobre essa técnica. Você pode dar a dica ao compilador que ele pode fazer um unrolling diretamente em seus programas em C.

```
int vetor[10];  
for(int i = 0; i < 10; i+=2){  
    vetor[i] = vetor[i]+1;  
    vetor[i+1] = vetor[i+1]+1;  
}  
//...
```

```
ori $s0,$zero,0 #inicio do vetor  
lui $s0,0x1000  
ori $s1,$zero,40 #contador regressivo
```

loop:

```
addi $s1,$s1,-4 #decrementa cont.  
addi $s2,$s0,$s1 #endereço efetivo  
lw $t0,0($s2)  
addi $t0,$t0,1  
sw $t0,0($s2)  
bne $s1,$zero, loop
```

```
ori $s0,$zero,0  
lui $s0,0x1000  
ori $s1,$zero,40  
nop  
nop  
nop
```

loop:

```
addi $s2,$s0,$s1  
nop  
lw $t0,-4($s2)  
addi $s1,$s1,-8  
lw $t1,-8($s3)  
addi $t0,$t0,1  
sw $t0,-4($s2)  
addi $t1,$t1,1  
sw $t1,-8($s3)  
bne $s1,$zero, loop  
nop
```

Loop Unrolling

Fazer um loop unrolling muito agressivo vai exigir muitos registradores extras.

Aumenta o *register pressure*.

Se os registradores acabarem, o compilador será obrigado a criar transações com a memória para salvar o estado dos registradores - *spill*.

```
int vetor[10];
for(int i = 0; i < 10; i+=2){
    vetor[i] = vetor[i]+1;
    vetor[i+1] = vetor[i+1]+1;
    vetor[i+2] = vetor[i+2]+1;
    vetor[i+3] = vetor[i+3]+1;
}
//...
```

VLIW

Vantagens do VLIW.

- + Enviando n instruções em um pacote, podemos aumentar a vazão em n vezes.
- + O hardware e o controle extras são relativamente simples.

Desvantagens?

VLIW

Vantagens do VLIW.

- + Enviando n instruções em um pacote, podemos aumentar a vazão em n vezes.
- + O hardware e o controle extras são relativamente simples.

Desvantagens.

- O compilador/programador terão muito mais trabalho.
- Nem sempre é possível montar um pacote completo.
No exemplo da aula, um pacote incompleto desperdiça 32 bits.
- Aumenta a pressão nas memórias.
Agora as memórias precisam servir mais bits de instrução, e armazenar mais bits a cada ciclo.
- Compatibilidade: e se criarmos outro processador VLIW, mas agora com pacotes de 4 instruções?

Despacho múltiplo estático

Processadores VLIW são de **despacho múltiplo estático**.

O compilador resolve quais instruções vão fazer parte de um pacote.

Em alguns processadores de despacho múltiplo estático, o compilador precisa resolver os hazards.

Inserir nops onde necessário.

Em outros, o processador pode resolver isso internamente com suas unidades de detecção de hazard e controle.

Mundo Real - Intel IA-64

Intel IA-64

Parceria entre Intel e HP

Processadores Itanium.



Mundo Real - Intel IA-64

Processadores VLIW com algumas melhorias e mais flexibilidade.

Por conta das melhorias, a Intel e HP chamaram a arquitetura de **EPIC**.

Explicitly Parallel Instruction Computer.

Mundo Real - Intel IA-64

- 128 registradores de 64 bits para uso geral.
- 128 registradores de 82 bits para ponto flutuante.
- Registradores para controle, mapeamento de memória, comunicação com S.O., ...

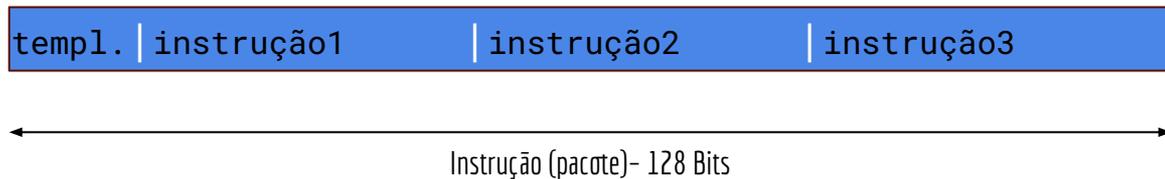
Mundo Real - Intel IA-64

As instruções IA-64 possuem 128 bits, chamadas de pacote.

5 bits de template para especificar detalhes do pacote (exemplo: tipo das instruções no pacote)

3 instruções de 41 bits.

Obs.: Essa é uma simplificação das instruções reais. Veja nos manuais Intel que as coisas são **muito** mais complicadas.



Unidades de Execução

Existem 5 unidades de execução possíveis.

Slot	Tipo	Descrição	Exemplos
Unidade I	A	ALU Inteiros	add, sub, or, ...
	I	Não ALU	mov, bit test, ...
Unidade M	A	ALU Inteiros	add, sub, or, ...
	M	Memória	load, store,...
Unidade F	F	Ponto Flutuante	instruções de fp.
Unidade B	B	Branch	branch, call, ...
L+X	L+X	Extendido	instruções com imediatos grandes, nop,...

Unidades de Execução

O campo de template da instrução indica a combinação de Unidades que serão usadas pelas instruções.

Existem 24 combinações de template válidas.

Exemplos:

Código Template	Inst0	Inst1	Inst2
0	M	I	I
12	M	F	I
16	M	I	B

Slot	Tipo	Descrição
Unidade I	A	ALU Inteiros
	I	Não ALU
Unidade M	A	ALU Inteiros
	M	Memória
Unidade F	F	Ponto Flutuante
Unidade B	B	Branch
L+X	L+X	Extendido

Mundo Real - Intel IA-64

A relativa flexibilidade dos pacotes (através do template) e a resolução de muitos itens em tempo de execução pelo próprio processador é uma característica do EPIC.

Por isso a Intel/HP “inventaram” a arquitetura EPIC, para diferenciar do VLIW.

Mundo Real - Intel IA-64

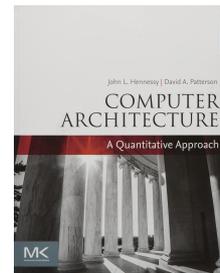
A relativa flexibilidade dos pacotes (através do template) e a resolução de muitos itens em tempo de execução pelo próprio processador é uma característica do EPIC.

Por isso a Intel/HP “inventaram” a arquitetura EPIC, para diferenciar do VLIW.

Mas você pode ver o Itanium como um processador VLIW com algumas características de um superescalar com despacho múltiplo dinâmico - veremos em outra aula.

Você pode pesquisar mais detalhes sobre o IA-64 no apêndice H do livro:

Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 6a ed. 2019.



Exercícios

1. Em 2019 a Intel anunciou o fim da produção dos processadores Itanium. Quais as desvantagens desses processadores, que levaram a Intel a descontinuí-los?
2. Considere o processador VLIW baseado em MIPS32 de exemplo da aula. Converta o trecho do programa a seguir para assembly desse processador. Considere que a variável `val` está no endereço `0x1000 0000`, e que o vetor já está alocado e inicia no endereço `0x1000 0004`.

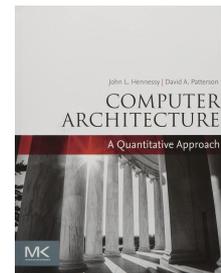
```
int vetor[10];
int val;
//...
for(int i = 1; i < 10; i++)
    vetor[i] = vetor[i] + vetor[i-1] + val;
//...
```

Referências

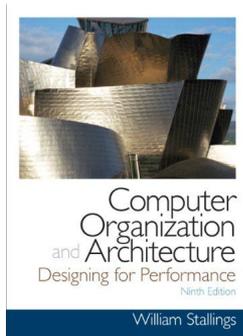
Patterson, Hennessy.
Arquitetura e Organização de
Computadores: A interface
hardware/software. 2014.



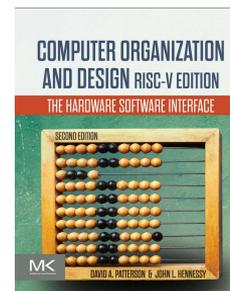
Hennessy, Patterson.
Arquitetura de Computadores:
uma abordagem quantitativa.
2019.



Stallings, W. Organização
de Arquitetura de
Computadores. 10a Ed.
2016.



Patterson, Hennessy.
Computer Organization and
Design RISC-V Edition: The
Hardware Software
Interface. 2020.



Licença

Esta obra está licenciada com uma Licença [Creative Commons Atribuição 4.0 Internacional](https://creativecommons.org/licenses/by/4.0/).

