Arquiteturas

Introdução a Arquiteturas de Computadores 3/3

Glauber De Bona

PCS - Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo

Setembro, 2020

Agenda

1 Chamadas

2 Endereçamento

3 Ferramentas

Chamadas

O que acontece em uma chamada de função?

1. Programa coloca os parâmetros em um local acessível;
2. Programa desvia o fluxo para o endereço da função;
3. Função obtém os recursos necessários para a execução;
4. Função executa sua tarefa;
5. Função coloca o resultado em um local acessível;
6. Função retorna o fluxo para o ponto de origem.

Chamadas

O LEGv8 possui suporte através de instruções com semântica especial:

1. Os registradores X0-X7 são reservados para os parâmetros ou resultados;
2. O registrador LR (X30) é reservado para guardar o endereço de retorno;
3. A instrução `BL Label` desvia o fluxo incondicionalmente para o endereço especificado pelo *label* **e guarda o endereço de retorno** no LR.
4. Para retornar da função, basta executar um `BR LR` (ou `BR X30`).

Chamadas

Há um registrador com propósito especial, usado para salvar os registradores (*spilling*):

1. O registrador é chamado de `SP` (*Stack Pointer*);
2. É um apontador para a cabeça da pilha na memória;
3. A pilha normalmente começa em uma posição alta da memória e cresce para baixo.

Chamadas (exemplo)

Vamos ver uma função:

```
1  int calcula (int g, int h, int i, int j){
2    int f;
3    f = (g + h) - (i + j);
4    return f;
5  }
```

Quando a função for chamada (`BL calcula`):

- os parâmetros estarão em `X0-X3`.
- o retorno (`f`) deve estar em `X0`.

Chamadas (exemplo)

Vamos usar três registradores para o cálculo, portanto precisamos salvá-los na pilha:

```
1   SUBI SP,SP,#24      // abre 3 espacos na pilha
2   STUR X10,[SP,#16]   // salva X10 na pilha
3   STUR X9,[SP,#8]     // salva X9 na pilha
4   STUR X19,[SP,#0]    // salva X19 na pilha
```

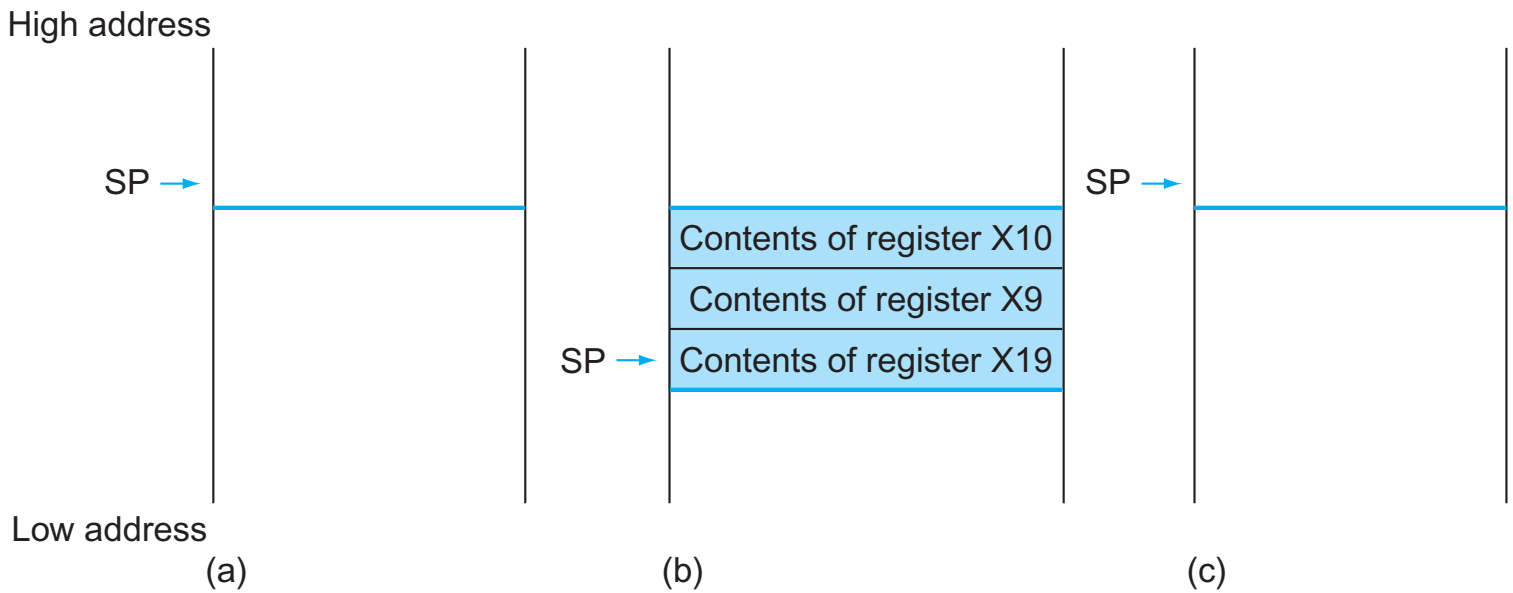
Chamadas (exemplo)

Vamos usar três registradores para o cálculo, portanto precisamos salvá-los na pilha:

```
1  SUBI SP,SP,#24      // abre 3 espacos na pilha
2  STUR X10,[SP,#16]   // salva X10 na pilha
3  STUR X9,[SP,#8]     // salva X9 na pilha
4  STUR X19,[SP,#0]    // salva X19 na pilha
```

Isto também é conhecido como **salvar o contexto**.

Chamadas (exemplo)



Chamadas (exemplo)

Agora faremos o cálculo da função:

```
1  ADD  X9, X0, X1     //  X9=X0+X1=g+h
2  ADD  X10, X2, X3    //  X10=X2+X3=i+j
3  SUB  X19, X9, X10   //  f=X19=X9-X10=(g+h)-(i+j)
```

E copiamos o resultado para o registrador de retorno:

```
1  ADD  X0, X19, XZR   //  X0=X19+0
```

Chamadas (exemplo)

Como cálculo feito, restauramos o contexto:

```
1   LDUR X19,[SP,#0]   // pop X19
2   LDUR X9,[SP,#8]    // pop X9
3   LDUR X10,[SP,#16]  // pop X10
4   ADDI SP,SP,#24     // ajustando a pilha
```

E retornamos:

```
1   BR LR              // desvio de retorno
```

Chamadas (exemplo)

Função completa:

```
1  calcula:  SUBI SP,SP,#24      // pushs
2            STUR X10,[SP,#16]
3            STUR X9,[SP,#8]
4            STUR X19,[SP,#0]
5            ADD X9,X0,X1        // calculo
6            ADD X10,X2,X3
7            SUB X19,X9,X10
8            ADD X0,X19,XZR      // resultado
9            LDUR X19,[SP.#0]    // pops
10           LDUR X9,[SP,#8]
11           LDUR X10,[SP,#16]
12           ADDI SP,SP,#24
13           BR LR               // retorno
```

O que uma Chamada deve Preservar

Para minimizar o *spilling*, por convenção, X9-X15 não precisam ser preservados nas chamadas.

Preserved	Not preserved
Saved registers: X19-X27	Temporary registers: X9-X15
Stack pointer register: X28(SP)	Argument/Result registers: X0-X7
Frame pointer register: X29(FP)	
Link Register (return address): X30(LR)	
Stack above the stack pointer	Stack below the stack pointer

Chamadas Aninhadas

- Funções que não chamam outras são chamadas de folhas.
- Não-folhas precisam empilhar o LR e os registradores temporários necessários.

```
1  fact:    SUBI SP,SP,#16
2           STUR LR,[SP,#8]  // salva LR
3           STUR X0,[SP,#0]  // salva o parametro
4           ...              // se X0=0, retorna X1=1
5           ...              // senao X0=X0-1
6           BL fact          // chamada recursiva
7           LDUR X0,[SP,#0]  // restaura parametro
8           LDUR X9,[SP,#8]  // restaura LR
9           ADDI SP,SP,#24
10          MUL  X1,X0,X1    // retorna n*fact(n-1)
11          BR LR            //
```

Chamadas (Exercício)

Qual o valor de X2 após a execução?

```
1          ADDI X0,XZR,#3
2          ADD X1,XZR,XZR
3          BL soma
4          ...
5  soma:   SUBS XZR,X0,XZR
6          B.LE s_sai
7          ADD X1,X1,X0
8          SUBI X0,X0,#1
9          B soma
10 s_sai:  ADD X2,X1,XZR
11         BR LR
```

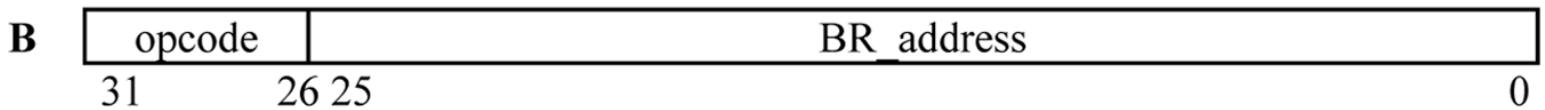
Endereçamento em Desvios

- Como condificar uma instrução `CBZ X0, label`?
- *opcode*, registrador, endereço para o salto.
  - Tirando o *opcode* e o registrador de teste, sobraram menos de 20 bits pro endereço!
  - **Problema**: Isso limitaria o tamanho dos programas a 2^{20}.
- Desvios condicionais normalmente vão para instruções próximas.
- **Solução**: Codificar um endereço relativo ao atual.
- *Program Counter (PC)* funciona como um registrador base, contendo o endereço da instrução atual.

Codificação - Formato CB

- Formato usado pelas instruções de salto condicional.
- Campos:
  - *opcode:* o código da instrução (CBZ, CBNZ, B.cond).
  - *COND_BR_address:* Endereço relativo em *words*.
    - Salto para *Program Counter*+*COND_BR_address**4
    - Salto até 1MiB de distância do PC.
  - *Rt:* O registrador para a condição (e.g. CBZ Rt, end).
    - A instrução B.cond especifica a condição em Rt.

Codificação - Formato B



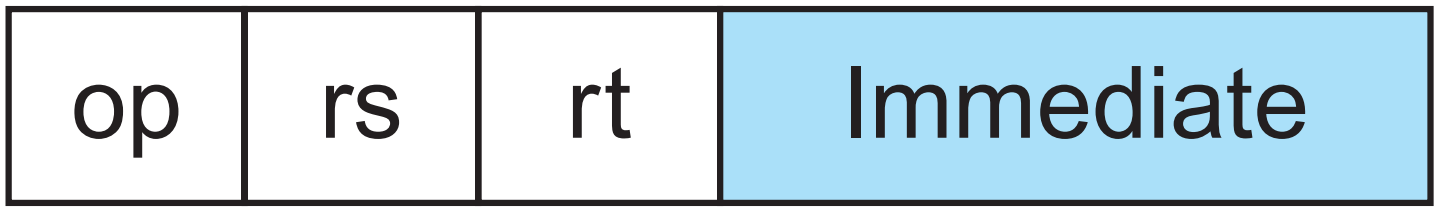
- Formato usado pela instrução de salto incondicional (B).
- Campos:
  - *opcode:* o código que representa a instrução.
  - *BR_address:* Endereço relativo de salto.
- **OBS**: BR usa o formato R, saltando para o endereço em Rd.

Modos de endereçamento

Vimos 4 modos diferentes de endereçamento no LEGv8:

- **Imediato**
- **Registrador**
- **Base**
- **Relativo**

Imediato

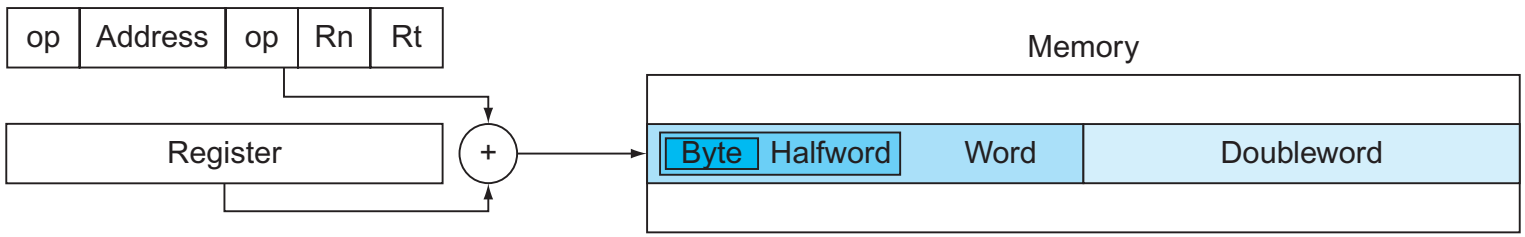


- Quando o valor a ser usado já está na instrução e não é necessário acesso externo.
- Instruções: todas de formato I e IW. Alguns campos de instruções de outros tipos também são considerados imediatos (e.g. *shamt* das instruções aritméticas).

Registrador

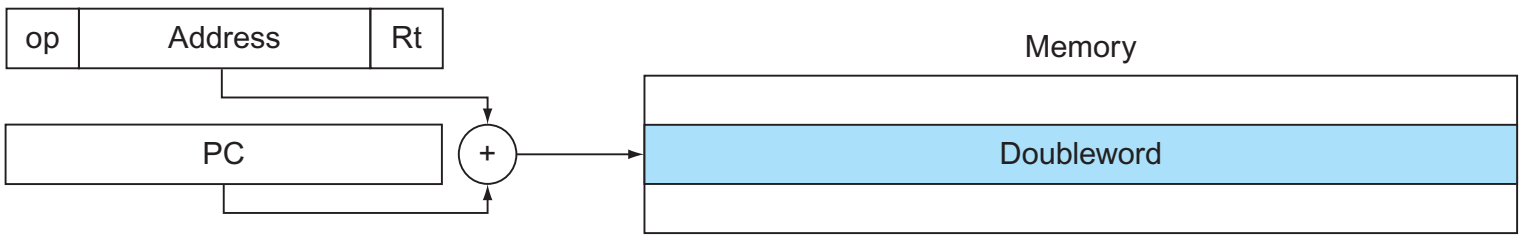
- Quando o valor na instrução (5 bits) aponta para um registrador no banco de registradores (sem acesso a memória).
- Instruções: todas de formato R. Sempre que um campo referencia um registrador, está usando este modo, portanto quase todas as instruções (exceto as do formato B) usam este modo.

Base



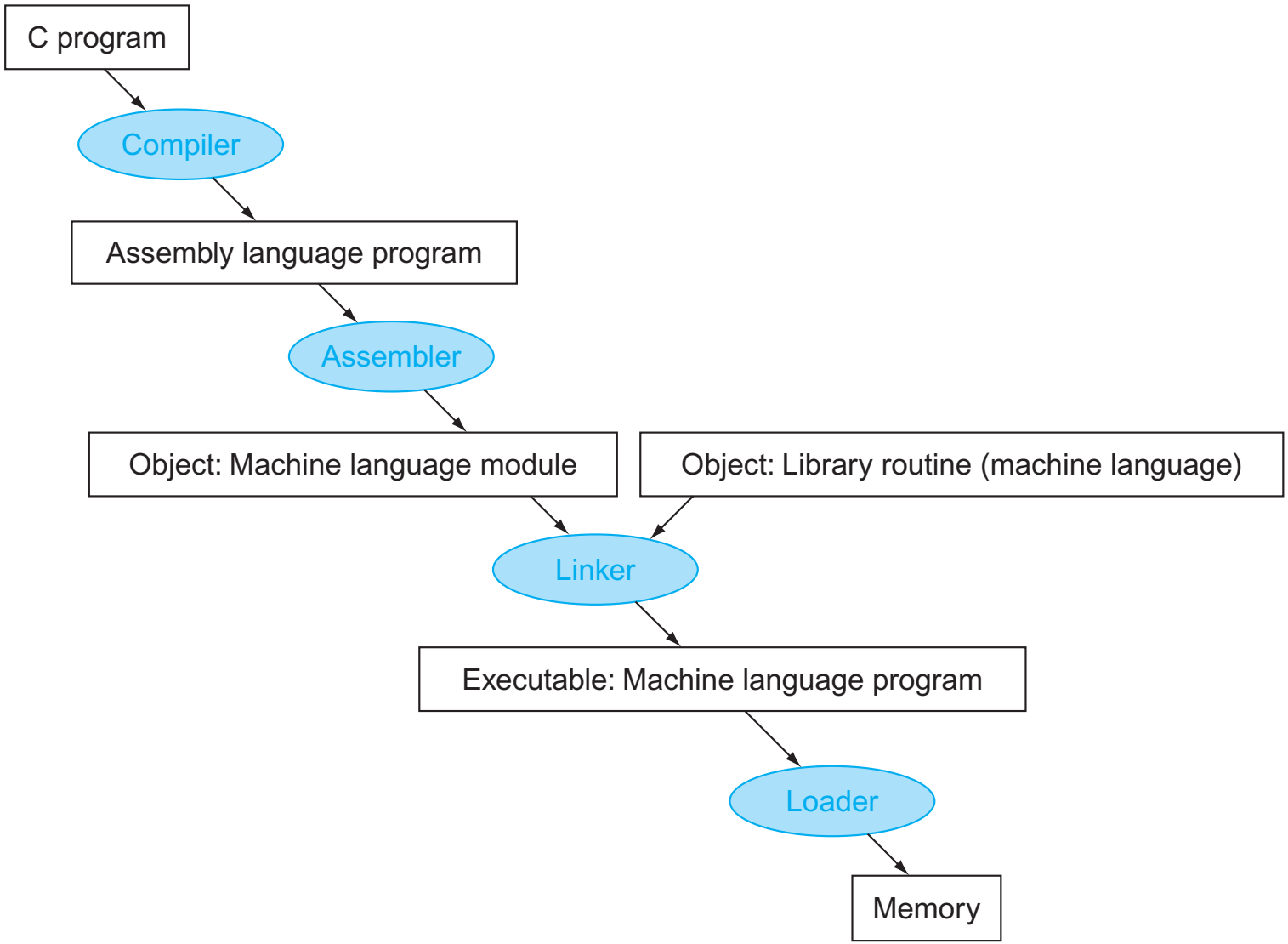
- Quando a instrução aponta um registrador e possui um valor imediato, que somados geram um apontador para uma palavra de memória.
- Instruções: todos `LD` e `ST`.

Relativo



- Quando a instrução tem um valor imediato, que somado ao PC aponta para uma palavra de memória.
- Também chamado de *PC-Relative*
- Instruções: todas instruções de desvio, exceto a BR que usa o modo de base com o imediato sempre em zero.

Ciclo de Compilação



Compilador

- Responsável por traduzir de um código de alto nível para *assembly*.
- A linguagem de alto nível é independente de arquitetura/máquina.
- O *assembly* é dependente de arquitetura.
- Uso de representação intermediária, onde podem ocorrer otimizações.
- Aloca registradores, variáveis e constantes.

Assembler

- Traduz de *assembly* para código de máquina.
- Normalmente não faz otimizações.
- Pode incluir reordenação e também instruções neutras para forçar bolhas no *pipeline*.
- O compilador pode usar pseudoinstruções, o assembler é responsável por traduzi-las para uma instrução real.
- Ex: `MOV X9, X10` -> `ORR X9, XZR, X10`

Linker

- Ordena o código executável e os dados simbólicos na memória.
- Determina o endereço dos dados e das instruções (*labels*).
- Atualiza referências para dados e instruções.
- Gera um código de máquina, porém executável.

Carregador (*Loader*)

- Lê o cabeçalho do executável e determina o espaço necessário em memória.
- Aloca um espaço de memória suficientemente grande para a execução.
- Copia o executável para a memória.
- Copia os parâmetros iniciais para os registradores (ou memória).
- Redireciona o fluxo de execução para o programa.

Obrigado!


Universidade de São Paulo
Departamento de Engenharia de Computação e Sistemas Digitais
PCS

Referências


D. Patterson and J. Hennessy.
Computer Organization and Design ARM Edition: The Hardware Software Interface.
The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2016.