



Bai3 Nguyen Minh Thuan B2207568

Kiến trúc máy tính (Trường Đại học Cần Thơ)



Scan to open on Studocu

BÀI TẬP 3: KIẾN TRÚC BỘ LỆNH

1) Hãy tìm mã hợp ngữ MIPS tương đương với chương trình C như sau:

a)

```
x = 5
y = x - 2
a = x * 4
b = y * 2
z = (x + a) - (y + b)
```

Biết rằng các biến x, y, a, b là các số nguyên 32 bit

b)

```
x = y + a[5]
```

Giả sử mỗi phần tử của mảng a là một word/từ nhớ 4 bytes.

a)

```
addi $s1, $zero, 5
addi $s2, $s1, -2
addi $s0, $zero, $zero
```

For:

```
    stl    $t1, $s3, 3      // if( i < n ) $s1 = 1; else $s0 = 0;
    bne    $t1, $zero, Exit // if ( i == 0 ) exit
    add    $s3, $s1, $s1    // a = x + x
    addi   $s1, $s1, 1      // i = i + 1
```

j For

Exit:

```
// sll $s3, $s1, 2 ( shift left logical 2 bit )
add $s4, $s2, $s2
// sll $s4, $s2, 1 ( shift left logical 1 bit )
add $s5, $s1, $s3
add $s6, $s2, $s4
sub $s7, $s5, $s6
```

b)

Giả sử mỗi phần tử mảng A là một word/từ nhớ 4 bytes

```
lw $t1, 20($t0)
add $s1, $s2, $t1
```

2) Hãy phân biệt sự khác nhau giữa Big End (Big Endian) và Little End (Little Endian). MIPS sử dụng Big End hay Little End?

Little endian và big endian là hai phương thức khác nhau để lưu trữ dữ liệu. Sự khác biệt của Little Endian và Big Endian khi lưu trữ chính là ở việc sắp xếp thứ tự các byte dữ liệu.

Trong cơ chế lưu trữ little endian (xuất phát từ "little-end" nghĩa kết thúc nhỏ hơn), byte cuối cùng trong biểu diễn nhị phân trên sẽ được ghi trước. Ví dụ 123456789_{ten} ghi theo kiểu little endian sẽ thành 15 cd 5b 07_(hexa)

Big endian (xuất phát từ "big-end") thì ngược lại, là cơ chế ghi dữ liệu theo thứ tự bình thường mà chúng ta vẫn dùng. được lưu trữ vẫn theo đúng thứ tự là 07 5b cd 15_(hexa)

MIPS lưu trữ dữ liệu theo dạng Big Endian, tức là byte cao sẽ được lưu ở địa chỉ thấp.

3) Cho 2 số x, y viết theo như sau:

x	y
4	-6
12	-9
19	-12
60	88

Hãy biểu diễn x, y theo nhị phân 8 bit có dấu bù 2 và thực hiện tính (x + y) theo hệ nhị phân

a) $4_{\text{ten}} + (-6)_{\text{ten}}$

$$6_{\text{ten}} = 0000\ 0110_{\text{two}}$$

$$\begin{array}{r}
 1111\ 1001_{\text{two}} \\
 + \quad 1 \\
 \hline
 1111\ 1010_{\text{two}} = -6_{\text{ten}}
 \end{array}$$

$$\begin{array}{r}
 0000\ 0100_{\text{two}} = 4_{\text{ten}} \\
 + \quad 1111\ 1010_{\text{two}} = -6_{\text{ten}} \\
 \hline
 1111\ 1110_{\text{two}} = -2_{\text{ten}}
 \end{array}$$

$$\begin{aligned}
 1111\ 1110_{\text{two}} &= -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 \\
 &= -128 + 64 + 32 + 16 + 8 + 4 + 2 = -128 + 40 + 80 + 6 = -2_{\text{ten}}
 \end{aligned}$$

b) $12_{\text{ten}} + (-9)_{\text{ten}}$

$$9_{\text{ten}} = 0000\ 1001_{\text{two}}$$

$$\begin{array}{r}
 1111\ 0110_{\text{two}} \\
 + \quad 1 \\
 \hline
 1111\ 0111_{\text{two}} = -9_{\text{ten}}
 \end{array}$$

$$\begin{array}{r}
 0000\ 1100_{\text{two}} = 12_{\text{ten}} \\
 + \quad 1111\ 0111_{\text{two}} = -9_{\text{ten}} \\
 \hline
 0000\ 0011_{\text{two}} = 3_{\text{ten}}
 \end{array}$$

$$c) 19_{ten} + (-12)_{ten}$$

$$12_{ten} = 0000\ 1100_{two}$$

$$\begin{array}{r}
 1111\ 0011_{two} \\
 + \quad 1 \\
 \hline
 1111\ 0100_{two} = -12_{ten}
 \end{array}$$

$$\begin{array}{r}
 0001\ 0011_{two} = 19_{ten} \\
 + \quad 1111\ 0100_{two} = -12_{ten} \\
 \hline
 0000\ 0111_{two} = 7_{ten}
 \end{array}$$

$$d) 60_{ten} + 88_{ten}$$

$$\begin{array}{r}
 0011\ 1100_{two} = 60_{ten} \\
 + \quad 0101\ 1000_{two} = 88_{ten} \\
 \hline
 1001\ 0100_{two} = 148_{ten}
 \end{array}$$

4) Liệt kê các định dạng lệnh MIPS

Hãy biểu diễn nhị phân lệnh: `add $t0,$s1,$s2`

	op	rs	rt	rd	shamt	funct
add	0	\$s1 = 17	\$s2 = 18	\$t0 = 8	0	32 _{ten}
	0000 0000	0001 0001	0001 0010	0000 1000	0000 0000	0010 0000

5) Mô tả các bước chuyển 1 chương trình C thành mã máy thực thi trên máy tính

Để chuyển đổi một chương trình C thành mã máy thực thi trên máy tính, ta cần thực hiện các bước sau:

1. Tiền xử lý (Preprocessing): Trình biên dịch (compiler) sẽ thực hiện tiền xử lý trên chương trình C. Trong quá trình này, các chỉ thị tiền xử lý (preprocessor directives) như `#include` và `#define` sẽ được xử lý. Ví dụ, các tệp tiêu đề được chèn vào chương trình và các macro được mở rộng.

2. Biên dịch (Compilation): Sau quá trình tiền xử lý, trình biên dịch sẽ chuyển đổi mã nguồn C thành mã gọi là mã hợp ngữ (assembly language). Mã hợp ngữ là một ngôn ngữ trung gian giữa ngôn ngữ C và mã máy. Nó sử dụng các hướng dẫn có thể hiểu được bởi bộ vi xử lý (CPU).

3. Hợp ngữ (Assembly): Bước tiếp theo là chuyển đổi mã hợp ngữ thành mã máy nhị phân. Quá trình này được gọi là hợp ngữ (assembly). Trình hợp ngữ (assembler) sẽ chuyển đổi mã hợp ngữ thành các lệnh và chỉ thị cụ thể cho kiến trúc bộ vi xử lý mà chương trình được thiết kế để chạy trên.

4. Liên kết (Linking): Nếu chương trình của bạn sử dụng các thư viện ngoài hoặc có nhiều tệp nguồn, bước liên kết sẽ được thực hiện. Trình liên kết (linker) sẽ kết hợp các tệp đối tượng (object files) đã được tạo ra từ quá trình biên dịch và hợp ngữ. Nó sẽ giải quyết các tham chiếu tới các hàm và biến trong các tệp khác nhau và tạo ra một tệp thực thi (executable file).

5. Tải chương trình (Loading): Sau khi quá trình liên kết hoàn tất, tệp thực thi sẽ được tải vào bộ nhớ để chạy. Trình tải (loader) sẽ đọc tệp thực thi và chuẩn bị môi trường thực thi cho chương trình. Nó sẽ gán các địa chỉ bộ nhớ cho các phần của chương trình và sắp xếp các tài nguyên cần thiết.

6. Thực thi chương trình: Cuối cùng, chương trình sẽ được thực thi trên bộ vi xử lý. CPU sẽ đọc các lệnh máy từ bộ nhớ, thực hiện các phép tính và thực hiện các chỉ thị của chương trình. Chương trình sẽ chạy và thực hiện các nhiệm vụ đã được định nghĩa trong mã nguồn C ban đầu.

Qua các bước trên, chương trình C của bạn sẽ được biên dịch và chuyển đổi thành mã máy thực thi có thể chạy trên máy tính.

6) Cho đoạn mã sau:

```
if (x < 0)
    h = 0;
else
    h = x;
```

với x, h được lưu trong \$s0, \$s1

Hãy dịch sang assembly MIPS đoạn mã trên

```
slt    $t0, $s0, $zero    // if(x<0) $t0 = 1 else $t0 = 0
beq    $t0, $zero, else   // if ($t0 == 0)
addi   $s1, $zero, 0      //      h = 0
j      exit
else:
    addi $s1, $s0, 0       // else h = x
exit
```

7) Các bước thực thi thủ tục

Để thực thi một thủ tục, chương trình máy tính phải tuân thủ theo sáu bước sau:

1. Đặt các tham số ở một nơi mà thủ tục có thể truy xuất được.
2. Chuyển quyền điều khiển cho thủ tục.
3. Yêu cầu tài nguyên lưu trữ cần thiết cho thủ tục đó.
4. Thực hiện các công việc (task).
5. Lưu kết quả ở một nơi mà chương trình có thể truy xuất được.
6. Trả điều khiển về vị trí mà thủ tục được gọi. Vì một thủ tục có thể được gọi từ nhiều vị trí trong một chương trình.

8) Cấu trúc thủ tục

main:

```
.....  
.....  
jal    abcde  
.....  
.....
```

abcde :

```
..... // Phần đầu  
.....//      - Khai báo kích thước cho stack  
.....//      - Lưu các thanh ghi cần thiết  
.....// Phần thân  
.....//      - Thực hiện chức năng yêu cầu ( có thể gọi các thủ tục khác..)  
.....// Phần cuối  
.....//      - Phục hồi các thanh ghi cần thiết  
.....//      - Xóa stack  
jr      $ra  
.....  
.....
```

Trong chương trình chính, khi thủ tục được gọi, con trỏ PC sẽ chuyển quyền điều khiển xuống vị trí của thủ tục; sau khi thủ tục được thực hiện xong, con trỏ PC sẽ chuyển về thực hiện lệnh ngay sau lệnh gọi thủ tục. Việc này được thực hiện nhờ vào cặp lệnh:

```
jal ten_thu_tuc  
jr $ra
```

(ten_thu_tuc là nhãn của lệnh đầu tiên trong thủ tục, cũng được xem là tên của thủ tục)

Để gọi thủ tục, dùng lệnh jal (Lệnh nhảy-và-liên kết: jump-and-link): Đầu tiên, địa chỉ của lệnh ngay sau lệnh jal phải được lưu lại để sau khi thủ tục thực hiện xong, con trỏ PC có thể chuyển về lại đây (thanh ghi dùng để lưu địa chỉ trả về là \$ra); sau khi địa chỉ trả về đã được lưu lại, con trỏ PC chuyển đến địa chỉ của lệnh đầu tiên của thủ tục để thực hiện. Vậy có hai công việc được thực hiện trong jal theo thứ tự:

```
jal    <address>          # $ra = PC + 4  
                                # PC = <address>
```

Sau khi thủ tục thực hiện xong, để trả về lại chương trình chính, dùng lệnh jr (jump regiser). Lệnh “ jr \$ra ” đặt ở cuối thủ tục thực hiện việc lấy giá trị đang chứa trong thanh ghi \$ra gán vào PC, giúp thanh ghi quay trở về thực hiện lệnh ngay sau lệnh gọi thủ tục này.

```
jr      $ra              # PC = $ra
```

Như vậy, mỗi hàm con/thủ tục được chia thành ba phần. Phần thân dùng để tính toán chức năng của thủ tục này, bắt buộc phải có. Phần đầu và phần cuối có thể có hoặc không, tùy từng yêu cầu của chương trình con