



การจัดองค์การคอมพิวเตอร์

พ4.6 ภาษาแอสเซมบลี (2/3)

31110321 Computer Organization

สำหรับนักศึกษาชั้นปีที่ 3 สาขาวิชาวิศวกรรมคอมพิวเตอร์

ทรงฤทธิ์ กิตติศรีวรพันธุ์

songrit@npu.ac.th

สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยนครพนม

Lecture plan

- 4.1 ภาษาเครื่อง
- 4.2 ส่วนประกอบพื้นฐาน
- 4.3 ระบบแอสกคคอมพิวเตอร์และภาษาเครื่อง
- 4.4 ภาษาเครื่องแอสกค
- 4.5 อินพุต / เอาท์พุต
- **4.6 การเขียนโปรแกรมสำหรับเครื่องแอสกค (2/3)**
- 4.7 ภาพรวมโปรเจกต์สัปดาห์4

Hack programming

- รีจิสเตอร์ และ หน่วยความจำ
- **Branching**
- Variables
- Iteration
- Pointers
- Input/output

Branching

- เจื่อนไขในการทำงานต่อไป
- ในแอสเซมบลี มีคำสั่งเดียวคือ goto

example:

- §

```
// Program: Signum.asm
// Computes: if R0>0
//           R1=1
//           else
//           R1=0
// Usage: put a value in RAM[0],
//        run and inspect RAM[1].

0
1      // D = RAM[0]
2
3
4
5
6
7
8
9
10
11
```

Branching

example:

```
0  // Program: Signum.asm
1  // Computes: if R0>0
2  //           R1=1
3  //           else
4  //           R1=0
5  // Usage: put a value in RAM[0],
6  //         run and inspect RAM[1].
7
8  @R0
9  D=M    // D = RAM[0]
10
11 @8
12 D;JGT  // If R0>0 goto 8
13
14 @R1
15 M=0    // RAM[1]=0
16 @10
17 0;JMP  // goto end
18
19 @R1
20 M=1    // R1=1
21
22 @10
23 0;JMP
```

cryptic code

“Instead of imagining that our main task as programmers is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.”

– Donald Knuth



แทนที่จะคำนึงว่าคอมพิวเตอร์ทำงานอย่างไร
ให้คำนึงถึงการอธิบายความต้องการที่เรา
ต้องการให้คอมพิวเตอร์เป็นเพื่อให้คนทั่วไป
เข้าใจ

Branching

example:

```
// Program: Signum.asm
// Computes: if R0>0
//           R1=1
//           else
//           R1=0
// Usage: put a value in RAM[0],
//        run and inspect RAM[1].
```

```
@R0
D=M    // D = RAM[0]
@POSITIVE
D;JGT  // If R0>0 goto 8
```

referring to a label

```
@R1
M=0    // RAM[1]=0
@10
0;JMP  // goto end
```

```
(POSITIVE)
@R1
M=1    // R1=1
```

declaring a label

```
(END)
@END
0;JMP
```

- แทนการเรียกตำแหน่งหน่วยความจำ
- ใช้ชื่อ แทนตำแหน่งหน่วยความจำ
 - @ชื่อ
 - (ชื่อ)

Branching

example:

```

// Program: Signum.asm
// Computes: if R0>0
//           R1=1
//           else
//           R1=0
// Usage: put a value in RAM[0],
//        run and inspect RAM[1].

0  @R0
1  D=M    // D = RAM[0]
2  @POSITIVE ← referring to a label
3  D;JGT  // If R0>0 goto 8
4
5  @R1
6  M=0    // RAM[1]=0
7  @10
8  0;JMP  // goto end
9
10 (POSITIVE) ← declaring a label
11 @R1
    M=1    // R1=1
12
13 (END)
14 @END
15 0;JMP

```

resolving
labels

- การกำหนด Label ไม่ได้สร้างคำสั่งใดๆ
- การอ้างถึง Label แต่ละครั้งจะมีการสร้างจุดคำสั่งซ้ำ

Memory

S:

0	@0
1	D=M
2	@8 // @POSITIVE
3	D;JGT
4	@1
5	M=0
6	@10 // @END
7	0;JMP
8	@1
9	M=1
10	@10 // @END
11	0;JMP
12	
13	
14	
15	
	⋮
32767	

Branching

example:

```
// Program: Signum.asm
// Computes: if R0>0
//           R1=1
//           else
//           R1=0
// Usage: put a value in RAM[0],
//        run and inspect RAM[1].
```

```
0  @R0
1  D=M    // D = RAM[0]
2  @POSITIVE ← referring to a label
3  D;JGT  // If R0>0 goto 8
4
5  @R1
6  M=0    // RAM[1]=0
7  @10
8  0;JMP  // goto end
9
10 (POSITIVE) ← declaring a label
11 @R1
12 M=1    // R1=1
13
14 (END)
15 @END
16 0;JMP
```

resolving
labels

Implications:

- Instruction numbers no longer needed in symbolic programming
- The symbolic code becomes *relocatable*.

Memory

0	@0
1	D=M
2	@8 // @POSITIVE
3	D;JGT
4	@1
5	M=0
6	@10 // @END
7	0;JMP
8	@1
9	M=1
10	@10 // @END
11	0;JMP
12	
13	
14	
15	
	⋮
32767	

Variables

```
// Program: Flip.asm  
// flips the values of  
// RAM[0] and RAM[1]  
  
// temp = R1  
// R1 = R0  
// R0 = temp
```

Variables

Variable usage example:

```
// Program: Flip.asm  
// flips the values of  
// RAM[0] and RAM[1]
```

```
// temp = R1  
// R1 = R0  
// R0 = temp
```

```
@R1
```

```
D=M
```

```
@temp
```

```
M=D // temp = R1
```

```
@R0
```

```
D=M
```

```
@R1
```

```
M=D // R1 = R0
```

```
@temp
```

```
D=M
```

```
@R0
```

```
M=D // R0 = temp
```

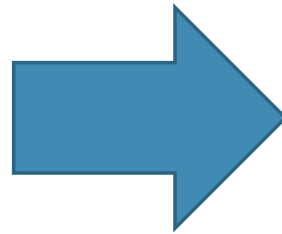
```
(END)
```

```
@END
```

```
0; JMP
```

symbol
used for the
first time

symbol
used again



Memory

0	@1
1	D=M
2	@16 // @temp
3	M=D
4	@0
5	D=M
6	@1
7	M=D
8	@16 // @temp
9	D=M
10	@0
11	M=D
12	@12
13	0; JMP
14	
15	
	⋮
32767	

Variable usage example:

```
// Program: Flip.asm
// flips the values of
// RAM[0] and RAM[1]

// temp = R1
// R1 = R0
// R0 = temp

    @R1
    D=M
    @temp
    M=D      // temp = R1

    @R0
    D=M
    @R1
    M=D      // R1 = R0

    @temp
    D=M
    @R0
    M=D      // R0 = temp

(END)
    @END
    0;JMP
```

resolving
symbols

Implications:

symbolic code is easy
to read and debug

Memory

0	@1
1	D=M
2	@16 // @temp
3	M=D
4	@0
5	D=M
6	@1
7	M=D
8	@16 // @temp
9	D=M
10	@0
11	M=D
12	@12
13	0;JMP
14	
15	
	⋮
32767	

arith.asm

pseudo code

```
// Computes RAM[1] = 1+2+ ... +RAM[0]
```

n = R0
i = 1
sum = 0

LOOP:

if i > n goto STOP

sum = sum + i

i = i + 1

goto LOOP

STOP:

R1 = sum

@R0

D=M

@n

M=D

@i

M=1

@sum

M=0

pseudo code

```
// Computes RAM[1] = 1+2+ ... +RAM[0]

n = R0
i = 1
sum = 0
...
```

assembly code

```
// Program: Sum1toN.asm
// Computes RAM[1] = 1+2+ ... +n
// Usage: put a number (n) in RAM[0]

@R0
D=M
@n
M=D // n = R0

@i
M=1 // i = 1

@sum
M=0 // sum = 0
...
```

Memory

0	@0
1	D=M
2	@16 // @n
3	M=D
4	@17 // @i
5	M=1
6	@18 // @sum
7	M=0
8	...
9	
10	
11	
12	
13	
14	
15	
	⋮
32767	

Variables are allocated to consecutive RAM locations from address 16 onward

assembly program

pseudo code

```
// Computes RAM[1] = 1+2+ ... +RAM[0]

n = R0
i = 1
sum = 0

LOOP:
  if i > n goto STOP
  sum = sum + i
  i = i + 1
  goto LOOP

STOP:
  R1 = sum
```

```
// Computes RAM[1] = 1+2+ ... +n
// Usage: put a number (n) in RAM[0]
@R0
D=M
@n
M=D // n = R0
@i
M=1 // i = 1
@sum
M=0 // sum = 0

(LLOOP)
@i
D=M
@n
D=D-M
@STOP
D;JGT // if i > n goto STOP
@sum
D=M
@i
D=D+M
@sum
M=D // sum = sum + i
@i
M=M+1 // i = i + 1
@LLOOP
0;JMP

(STOP)
@sum
D=M
@R1
M=D // RAM[1] = sum

(END)
@END
0;JMP
```

assembly program

```
// Computes RAM[1] = 1+2+ ... +n
// Usage: put a number (n) in RAM[0]
@R0
D=M
@n
M=D // n = R0
@i
M=1 // i = 1
@sum
M=0 // sum = 0
(LLOOP)
@i
D=M
@n
D=D-M
@STOP
D;JGT // if i > n goto STOP
@sum
D=M
@i
D=D+M
@sum
M=D // sum = sum + i
@i
M=M+1 // i = i + 1
@LOOP
0;JMP
(STOP)
@sum
D=M
@R1
M=D // RAM[1] = sum
(END)
@END
0;JMP
```

เลขที่ #

	0
RAM[0] :	3
n:	3
i:	1
sum:	0

assembly program

```
// Computes RAM[1] = 1+2+ ... +n
// Usage: put a number (n) in RAM[0]
@R0
D=M
@n
M=D // n = R0
@i
M=1 // i = 1
@sum
M=0 // sum = 0
(LABEL)
@i
D=M
@n
D=D-M
@STOP
D;JGT // if i > n goto STOP
@sum
D=M
@i
D=D+M
@sum
M=D // sum = sum + i
@i
M=M+1 // i = i + 1
@LABEL
0;JMP
(STOP)
@sum
D=M
@R1
M=D // RAM[1] = sum
(END)
@END
0;JMP
```

• แนวทาง

- **ออกแบบ** โปรแกรมโดยใช้ pseudo code
- **เขียน** โปรแกรมด้วยแอสเซมบลี
- **ทดสอบ** โปรแกรม (ในกระดาษ) เขียนตัวแปรแล้วรันทีละบรรทัด ตรวจสอบการเปลี่ยนแปลง

Lecture plan

- 4.1 ภาษาเครื่อง
- 4.2 ส่วนประกอบพื้นฐาน
- 4.3 ระบบแอสกคคอมพิวเตอร์และภาษาเครื่อง
- 4.4 ภาษาเครื่องแอสกค
- 4.5 อินพุต / เอาท์พุต
- **4.6 การเขียนโปรแกรมสำหรับเครื่องแอสกค (2/3)**
- 4.7 ภาพรวมโปรเจ็ค