The Elements of Computing Systems

Chapter 6: Assembler

kekeho

2022/05/19 kumo+bcali

6.1 Background

Introduction

- **Binary**: Instructions that hardware can execute directly
  - 1111 1010 1001 0000
- **Assembly**:
  - D=D-A
- **Assembler**: Convert Hack assembly to binary
  - D=D-A → Assembler → 1111 1010 1001 0000

6.1 Background

Symbol

- **Variables**: Assembler assigned memory address automatically
  - From 1024, allocated in order of appearance
- **Labels**: Programmer can mark various positions in the program with symbols.
  - Assign line numbers for label occurrences

```
00  i=1
01  sum = 0
    LOOP:
02    if i=101 goto END
03    sum=sum+i
04    i=i+1
05    goto LOOP
    END:
06    goto END
```

End of program with infinite loop

```
00  M[1024] = 1
01  M[1025] = 0
02  if M[1024]=101 goto 6
03  M[1025]=M[1025]+M[1026]
04  M[1024]=M[1024]+1
05  goto 2
06  goto 6
```

6.1 Background

Assembler

1. Parse the symbolic command into its underlying fields.
2. For each field, generate the corresponding bits in the machine language.
3. Replace all symbolic references (if any) with numeric addresses of memory locations.
4. Assemble the binary codes into a complete machine instruction.

6.2 Hack Assembly-to-Binary Translation Specification

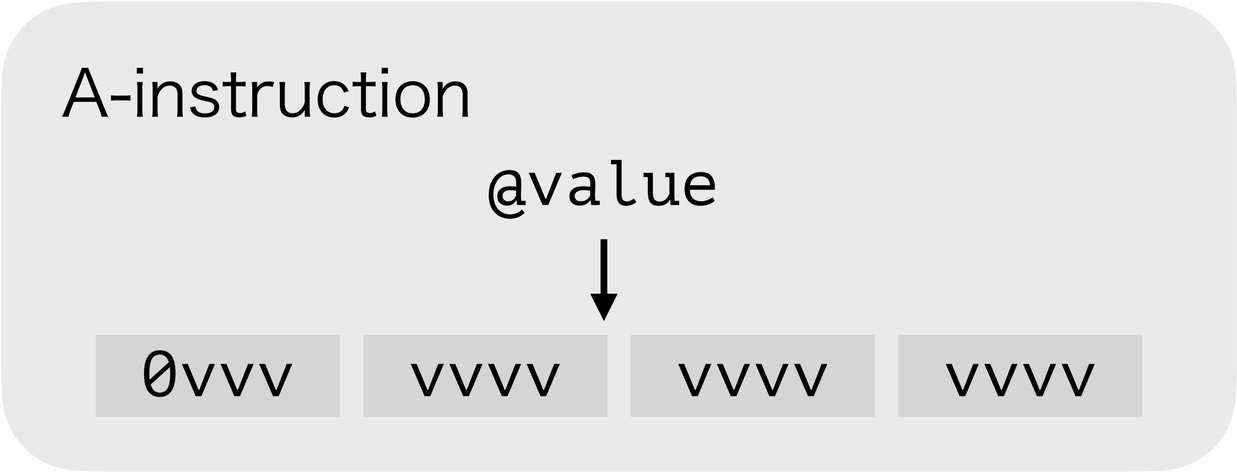
Syntax Conventions and File Formats

- **Binary file (.hack)**
  - Each line contains 16 ASCII characters of 0 or 1
  - Program line number and memory address are the same.
- **Assembly file (.asm)**
  - Instruction: A-instruction or C-instruction
  - Symbol
  - Comment (//)

6.2 Hack Assembly-to-Binary Translation Specification

Instruction

- **A-instruction**: Set value to A-register



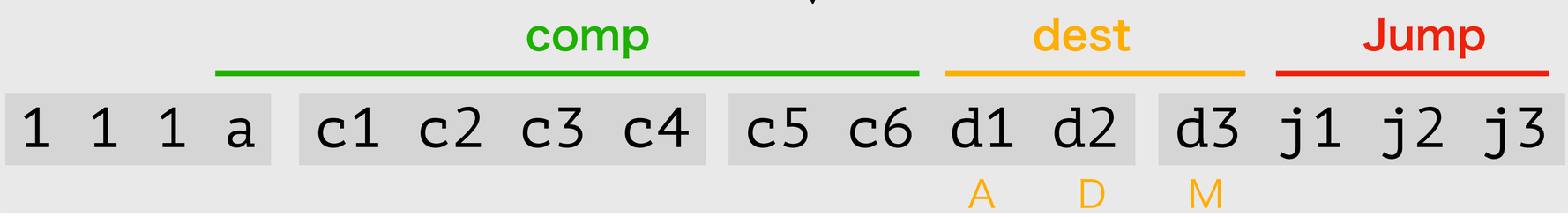
6.2 Hack Assembly-to-Binary Translation Specification

Instruction

- **C-instruction:** Compute instruction

C-instruction

`dest=comp:jump`



6.2 Hack Assembly-to-Binary Translation Specification

Instruction

- **C-instruction:** Compute instruction

jump	j1	j2	j3
null	0	0	0
JGT	0	0	1
JEQ	0	1	0
JGE	0	1	1
JLT	1	0	0
JNE	1	0	1
JLE	1	1	0
JMP	1	1	1

comp (when a=0)	c1	c2	c3	c4	c5	c6	*comp* (when a=1)
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

6.2 Hack Assembly-to-Binary Translation Specification

Pre-defined Symbol

Label	*RAM address*	*(hexa)*
SP	0	0x0000
LCL	1	0x0001
ARG	2	0x0002
THIS	3	0x0003
THAT	4	0x0004
R0-R15	0-15	0x0000-f
SCREEN	16384	0x4000
KBD	24576	0x6000

6.3 Implementation

General Policy

First step

Generate Symbol Table

Symbol Table

Symbol	Value
OUTPUT_FIRST	10
OUTPUT_D	12
INFINITE_LOOP	14

```
00      @R0
01      D=M
02      @R1
03      D=D-M
04      @OUTPUT_FIRST
05      D;JGT
06      @R1
07      D=M
08      @OUTPUT_D
09      0;JMP
    (OUTPUT_FIRST)
10      @R0
11      D=M
    (OUTPUT_D)
12      @R2
13      M=D
    (INFINITE_LOOP)
14      @INFINITE_LOOP
15      0;JMP
```

Second Step

Translate to binary

```
00  0000000000000000
01  1111110000010000
02  0000000000000001
03  1111010011010000
04  0000000000001010
05  1110001100000001
06  0000000000000001
07  1111110000010000
08  0000000000001100
09  1110101010000111
10  0000000000000000
11  1111110000010000
12  0000000000000010
13  1110001100001000
14  0000000000001110
15  1110101010000111
```