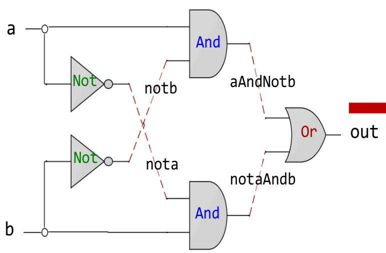
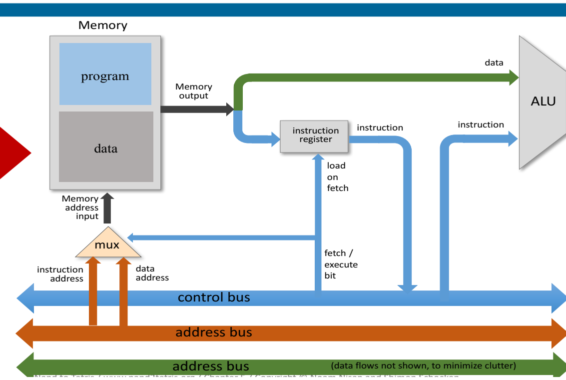
Digital Logic Design




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Term Project

Design of Hack Assembler

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- What is an Assembler?
- How an Assembler works?
- Hack Machine Language Specification
- Demo of Built-in Hack Assembler
- Design of Hack Assembler (w/o Symbols)
- Design of Hack Assembler (with Symbols)
- Hack Assembler Implementation in C/C++
- Executing Hack Machine Code
  - Hack Computer Chip in h/w Simulator
  - CPU Emulator

What is an assembler?
&
How an Assembler works?

What is an Assembler?

An assembler is a program that takes as input, a stream of assembly instructions and generates as output a stream of equivalent binary instructions. The resulting code can be loaded as is into the computer's memory and executed by the processor

```
   // Program: swap.asm
   // Usage: put values in RAM[0], RAM[1]
   //swap the values of RAM[0] and RAM[1]
1  @R1
2  D=M
3  @temp
4  M=D

5  @R0
6  D=M
7  @R1
8  M=D

9  @temp
10 D=M
11 @R0
12 M=D
   (END)
13 @END
14 0;JMP
```

Assembler

Translate

swap.hack

```
1  0000000000000001
2  1111110000010000
3  0000000000010000
4  1110001100001000
5  0000000000000000
6  1111110000010000
7  0000000000000001
8  1110001100001000
9  0000000000010000
10 1111110000010000
11 0000000000000000
12 1110001100001000
13 0000000000001100
14 1110101010000111
```

Load in Hack Computer

Execute

Where does the Assembler Program Runs?

```
@R1
D=M
@temp
M=D
@R0
D=M
@R1
M=D
@temp
D=M
@R0
M=D
(END)
@END
0;JMP
```

Assembly Language Program
`swap.asm`

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
0000000000000000
1111110000010000
0000000000000001
1110001100001000
0000000000010000
1111110000010000
0000000000000000
1110001100001000
0000000000001100
1110101010000111
```

Machine Language Program
`swap.hack`

Assembler

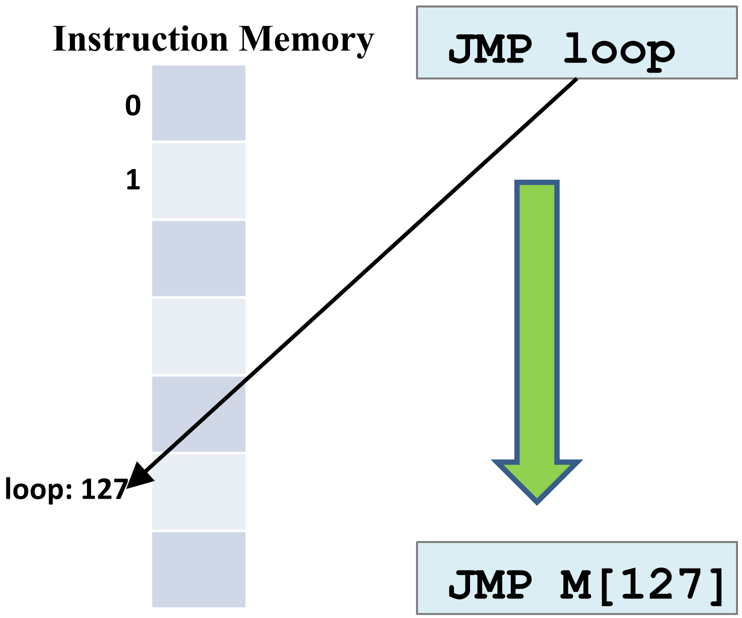
Your Home PC

Your Hack Computer

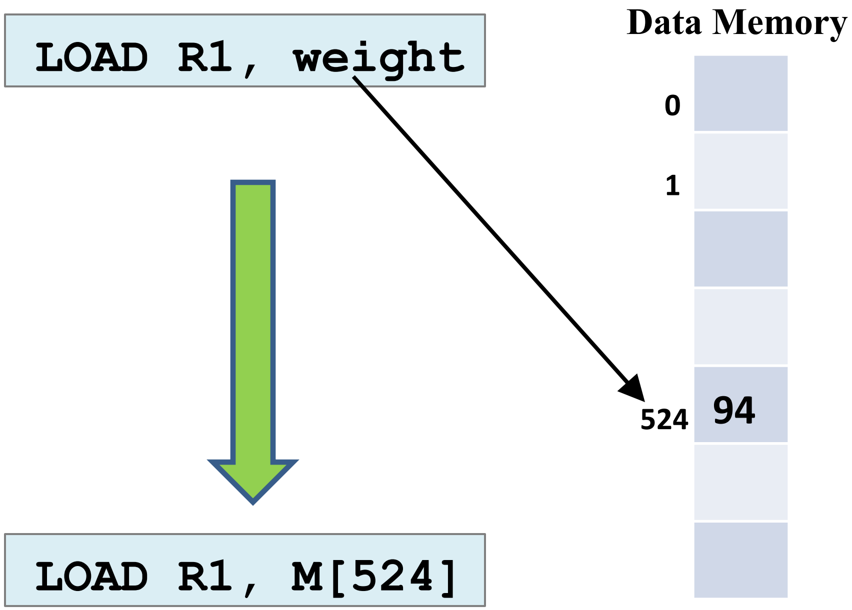
Symbols in Assembly Language

- Assembly Instructions can refer to memory locations (addresses) using either constants or symbols. Other than the predefined/build-in symbols, an assembly programmer can use user-defined symbols in following two ways:

Label Symbols:



Variable Symbols:



How can an Assembler Resolves these Symbols

Code with Symbols

```
   // Computes sum = 1+2+….+100
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
```

Resolve Symbols →

Code with Symbols Resolved

```
00  M[1024]=1
01  M[1025]=0
02  if  M[1024]==101 goto ?
```

Two Pass Assembler and Symbol Table

- A Two Pass Assembler is an assembler that goes through the source file twice, in the first pass it creates symbol table for that file and in the second pass it resolves all the symbol references and generate the appropriate machine code
- A symbol table is a data structure used by an assembler/compiler to look-up and resolve symbolic names with their corresponding memory addresses

Code with Symbols

```
   // Computes sum = 1+2+….+100
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
```

Translate

Symbol Table

i	1024
sum	1025
loop	2
end	6

Assuming that variables are allocated to Memory[1024] onward

Code with Symbols Resolved

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

Assuming that each symbolic command is translated into one word in memory

How an Assembler Work?

Read next assembly language instruction from file:

- Parsing: Break symbolic instruction into its underlying fields
- Code Generation: For each field, generate the corresponding bits in the machine language
- Symbol Handling: Replace all symbolic references with numeric addresses of memory locations
- Assembly: Combine the binary codes into a complete machine instruction and write this machine language instruction to output file

Repeat, Until End of file is reached

How an Assembler Work? (…)

Read next assembly language instruction from file:

```
//This is a comment
Load R1,35
```

Ignore comments , blank lines and white spaces

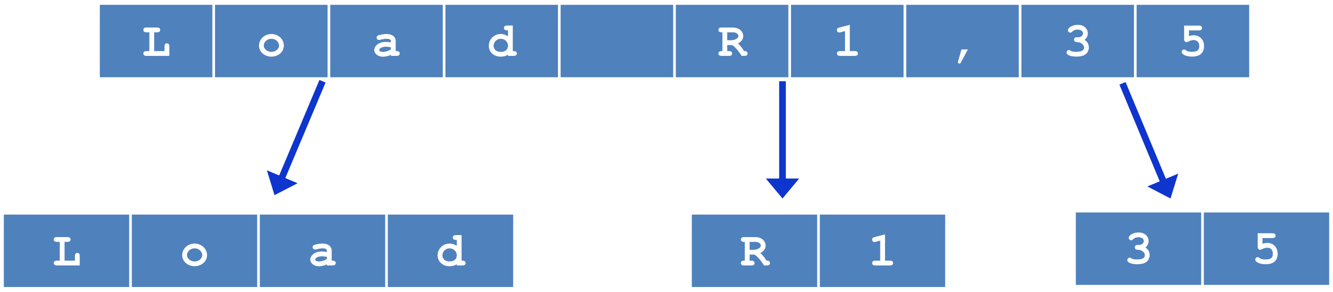
Read assembly instruction into an array of characters

L	o	a	d		R	1	,	3	5

How an Assembler Work? (…)

Read next assembly language instruction from file:

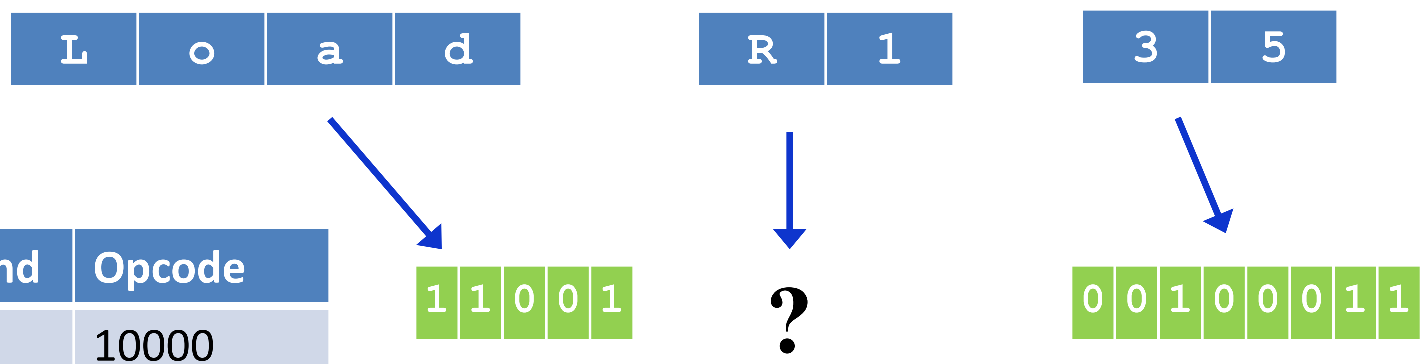
- Parsing: Break symbolic instruction into its underlying fields



How an Assembler Work? (…)

Read next assembly language instruction from file:

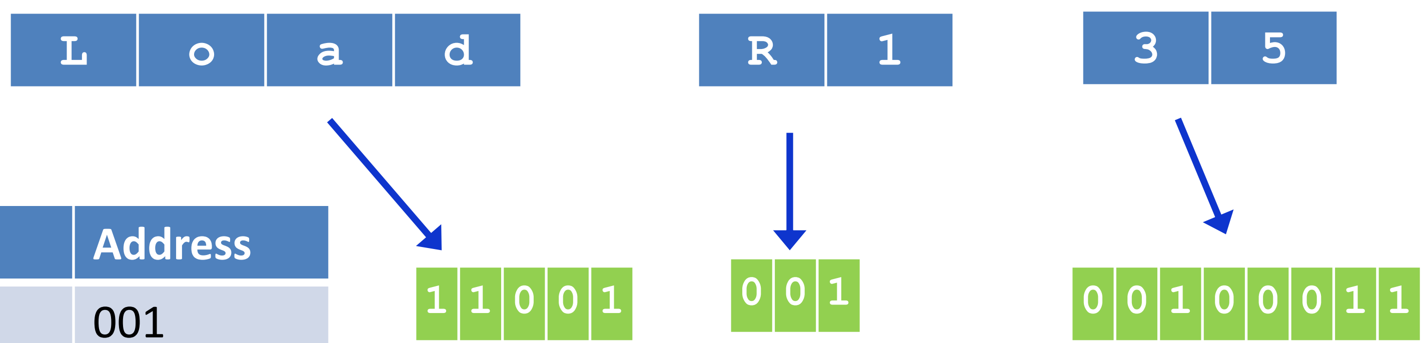
- ❑ Parsing: Break symbolic instruction into its underlying fields
- ❑ Code Generation: For each field, generate the corresponding bits in the machine language



Command	Opcode
Add	10000
Sub	10001
---	---
Load	11001

Read next assembly language instruction from file:

- ❑ Parsing: Break symbolic instruction into its underlying fields
- ❑ Code Generation: For each field, generate the corresponding bits in the machine language
- ❑ Symbol Handling: Replace all symbolic references with numeric addresses of memory locations

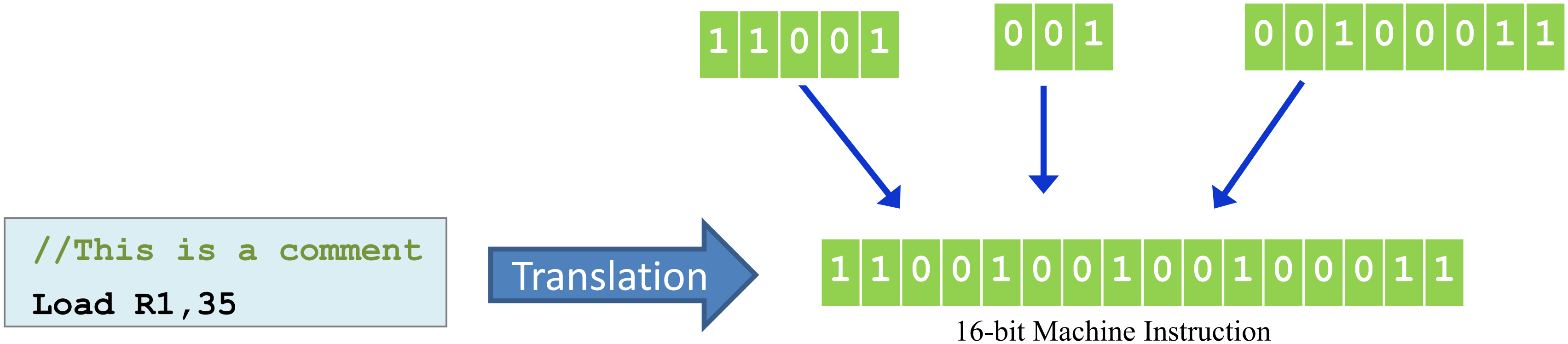


Symbol	Address
R1	001
R2	010
---	---

How an Assembler Work? (…)

Read next assembly language instruction from file:

- ❑ Parsing: Break symbolic instruction into its underlying fields
- ❑ Code Generation: For each field, generate the corresponding bits in the machine language
- ❑ Symbol Handling: Replace all symbolic references with numeric addresses of memory locations
- ❑ Assembly: Combine the binary codes into a complete machine instruction and write this machine language instruction to output file



Note: The output is written in a file as per the specification of the file format of machine language which may be a binary format, or a text format that the target computer understand as an executable file format

Recap:
Hack Machine Language Specification

Hack Language Specification: A-Instruction

The A-instruction is used to set the A register to a 15 bit value

Syntax:

```
@ value
```

Translation to binary:

- If *value* is a decimal constant, generate the equivalent binary constant
- If *value* is a symbol, resolve it

Symbolic Code:

```
@23
```

Translate

Machine Code:

```
0000 0000 0001 0111
```

opcode signifying an A-instruction

Hack Language Specification: C-Instruction

`dest= comp ;jump`

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3

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

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

D=D+A → Translate → 1 11 0000010 010 000

Hack Language Specification: C-Instruction

dest= comp ;jump

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3

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

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

```
@17
D-1;JEQ
```

Translate

```
0 000000000010001
1 11 0001110 000 010
```

Hack Language Specification: Symbols

Pre-defined symbols:

Symbol	Value	Symbol	Value
R0	0	SP	0
R1	1	LCL	1
R2	2	ARG	2
...	...	THIS	3
R15	15	THAT	4
SCREEN	16384		
KBD	24576		

```
// Program: swap.asm
// Usage: put values in RAM[0], RAM[1]
//swap the values of RAM[0] and RAM[1]
@R1
D=M
@temp
M=D

@R0
D=M
@R1
M=D

@temp
D=M
@R0
M=D
(END)
@END
0;JMP
```

Label Symbols:

(LABELNAME) **@LABELNAME**

Variable Symbols:

@variablename

Built-in Hack Assembler

The Translator’s Challenge (Overview)

Hack Assembly Code

(Source Language)

```
   // Program: swap.asm
   // Usage: put values in RAM[0], RAM[1]
   //swap the values of RAM[0] and RAM[1]
1   @R1
2   D=M
3   @temp
4   M=D

5   @R0
6   D=M
7   @R1
8   M=D

9   @temp
10  D=M
11  @R0
12  M=D
    (END)
13  @END
14  0;JMP
```

Hack Assembler

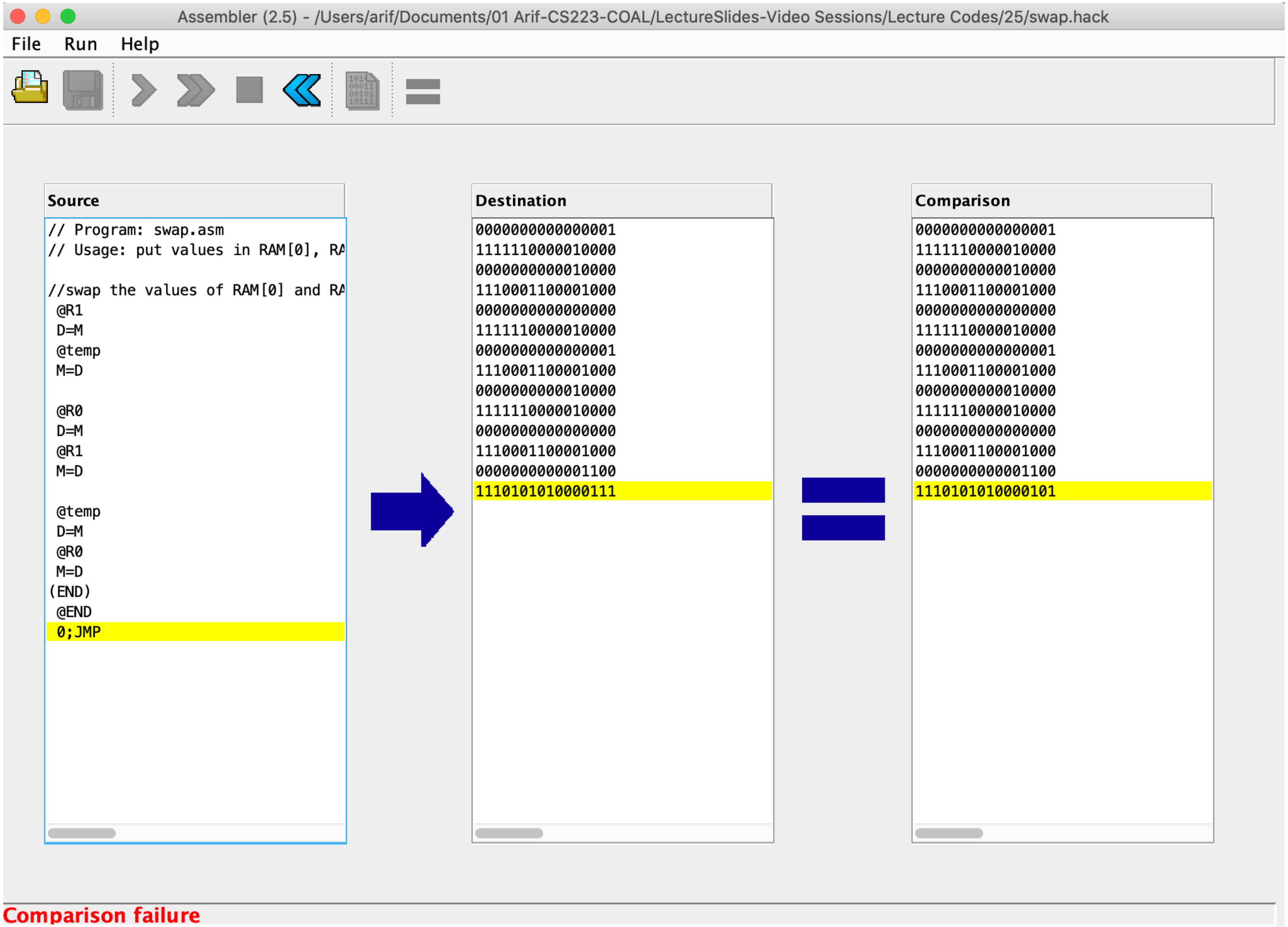
What are the rules of the game?

Hack Binary Code

(Target Language)

```
1   0000000000000001
2   1111110000010000
3   0000000000010000
4   1110001100001000
    Ignore this line
5   0000000000000000
6   1111110000010000
7   0000000000000001
8   1110001100001000
    Ignore this line
9   0000000000010000
10  1111110000010000
11  0000000000000000
12  1110001100001000
    Ignore this line
13  0000000000001100
14  1110101010000111
```

The Translator's Challenge (Overview)



Hack Assembler Tool (GUI)

Executing Hack Machine Code

- ✓ Hack Computer Chip in h/w Simulator
- ✓ CPU Emulator

Hack Assembler
w/o Symbols

The Hack Language: Translator's Perspective

Hack Assembly Program

```
// Computes RAM[1] = 1 + 2 + 3 … + RAM[0]
  @i
  M=1    //i = 1
  @sum
  M=0    //sum = 0
(LOOP)
  @i
  D=M
  @R0
  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
```

Assembly Program Elements:

White space

- Empty lines / indentation
- Line comments
- In-line comments

Ignore

The Hack Language: Translator's Perspective

Hack Assembly Program

```
   @i
   M=1
   @sum
   M=0
(LOOP)
   @i
   D=M
   @R0
   D=D-M
   @STOP
   D;JGT
   @sum
   D=M
   @i
   D=D+M
   @sum
   M=D
   @i
   M=M+1
   @LOOP
   0;JMP
(STOP)
   @sum
   D=M
   @R1
   M=D
(END)
   @END
   0;JMP
```

Assembly Program Elements:

White space

- Empty lines / indentation
- Line comments
- In-line comments

Ignore

Symbols

- Built-in Symbols
- Labels
- Variables

Assume the assembly programmer do not use symbols

The Hack Language: Translator's Perspective

Hack Assembly Program

```
@16
M=1
@17
M=0
@16
D=M
@0
D=D-M
@20
D;JGT
@17
D=M
@16
D=D+M
@17
M=D
@16
M=M+1
@4
0;JMP
@17
D=M
@1
M=D
@24
0;JMP
```

Assembly Program Elements:

White space

- Empty lines / indentation
- Line comments
- In-line comments

Ignore

Symbols

- Built-in Symbols
- Labels
- Variables

Assume the assembly programmer do not use symbols

Instructions

- A-instructions
- C-instructions

Translate

The Hack Language: Translator's Perspective

Hack Assembly Program (without Symbols)

0	@16
1	M=1
2	@17
3	M=0
4	@16
5	D=M
6	@0
7	D=D-M
8	@20
9	D;JGT
10	@17
11	D=M
12	@16
13	D=D+M
14	@17
15	M=D
16	@16
17	M=M+1
18	@4
19	0;JMP
20	@17
21	D=M
22	@1
23	M=D
24	@24
25	0;JMP

Assembler
for symbol-less Hack programs

Hack Binary Code

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

For each instruction

- **Parsing:** Break symbolic instruction into its underlying fields
- **Code Generation:**
  - A-instruction: translate the decimal value into a binary value
  - C-instruction: for each field in the instruction, generate the corresponding binary code
- **Symbol Handling:** No symbols exist
- **Assembly:** Combine the binary codes into 16-bit instruction

The Hack Language: Translator's Perspective

Hack Assembly Program (without Symbols)

0	@16
1	M=1
2	@17
3	M=0
4	@16
5	D=M
6	@0
7	D=D-M
8	@20
9	D;JGT
10	@17
11	D=M
12	@16
13	D=D+M
14	@17
15	M=D
16	@16
17	M=M+1
18	@4
19	0;JMP
20	@17
21	D=M
22	@1
23	M=D
24	@24
25	0;JMP

Assembler
for symbol-less Hack programs

Hack Binary Code

0	0000000000010000
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

For each instruction

- **Parsing:** Break symbolic instruction into its underlying fields
- **Code Generation:**
  - A-instruction: translate the decimal value into a binary value
  - C-instruction: for each field in the instruction, generate the corresponding binary code
- **Symbol Handling:** No symbols exist
- **Assembly:** Combine the binary codes into 16-bit instruction

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3

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

dest	d1	d2	d3
null	0	0	0
M	0	0	1
D	0	1	0
MD	0	1	1
A	1	0	0
AM	1	0	1
AD	1	1	0
AMD	1	1	1

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

The Hack Language: Translator's Perspective

Hack Assembly Program (without Symbols)

0	@16
1	M=1
2	@17
3	M=0
4	@16
5	D=M
6	@0
7	D=D-M
8	@20
9	D;JGT
10	@17
11	D=M
12	@16
13	D=D+M
14	@17
15	M=D
16	@16
17	M=M+1
18	@4
19	0;JMP
20	@17
21	D=M
22	@1
23	M=D
24	@24
25	0;JMP

Assembler
for symbol-less Hack programs

Hack Binary Code

0	0000000000010000
1	1110111111001000
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

For each instruction

- **Parsing:** Break symbolic instruction into its underlying fields
- **Code Generation:**
  - A-instruction: translate the decimal value into a binary value
  - C-instruction: for each field in the instruction, generate the corresponding binary code
- **Symbol Handling:** No symbols exist
- **Assembly:** Combine the binary codes into 16-bit instruction

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3

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

dest	d1	d2	d3
null	0	0	0
M	0	0	1
D	0	1	0
MD	0	1	1
A	1	0	0
AM	1	0	1
AD	1	1	0
AMD	1	1	1

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

The Hack Language: Translator's Perspective

Hack Assembly Program (without Symbols)

0	@16
1	M=1
2	@17
3	M=0
4	@16
5	D=M
6	@0
7	D=D-M
8	@20
9	D;JGT
10	@17
11	D=M
12	@16
13	D=D+M
14	@17
15	M=D
16	@16
17	M=M+1
18	@4
19	0;JMP
20	@17
21	D=M
22	@1
23	M=D
24	@24
25	0;JMP

Assembler
for symbol-less Hack programs

For each instruction

- **Parsing:** Break symbolic instruction into its underlying fields
- **Code Generation:**
  - A-instruction: translate the decimal value into a binary value
  - C-instruction: for each field in the instruction, generate the corresponding binary code
- **Symbol Handling:** No symbols exist
- **Assembly:** Combine the binary codes into 16-bit instruction

1	11	a c1 c2 c3 c4 c5 c6	d1 d2 d3	j1 j2 j3

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

dest	d1	d2	d3
null	0	0	0
M	0	0	1
D	0	1	0
MD	0	1	1
A	1	0	0
AM	1	0	1
AD	1	1	0
AMD	1	1	1

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

Hack Binary Code

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	0000000000010000
13	1111000010010000
14	0000000000010001
15	1110001100001000
16	0000000000010000
17	1111110111001000
18	0000000000000100
19	1110101010000111
20	0000000000010001
21	1111110000010000
22	0000000000000001
23	1110001100001000
24	0000000000011000
25	1110101010000111

Hack Assembler with Symbols

Hack Assembler

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Assembler
For Hack programs with symbols

Hack Binary Code

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	0000000000010000
13	1111000010010000
14	0000000000010001
15	1110001100001000
16	0000000000010000
17	1111110111001000
18	0000000000000100
19	1110101010000111
20	0000000000010001
21	1111110000010000
22	0000000000000001
23	1110001100001000
24	0000000000011000
25	1110101010000111

Challenges:

Handling…

- ✓ White space
- ✓ Instructions
- • Symbols

```
0   @i
1   M=1
2   @sum
3   M=0
    (LOOP)
4   @i
5   D=M
6   @R0
7   D=D-M
8   @STOP
9   D;JGT
10  @sum
11  D=M
12  @i
13  D=D+M
14  @sum
15  M=D
16  @i
17  M=M+1
18  @LOOP
19  0;JMP
    (STOP)
20  @sum
21  D=M
22  @R1
23  M=D
    (END)
24  @END
25  0;JMP
```

Handling Symbols

Symbols:

Pre-defined symbols:

Represent special memory locations

(**R0**, **R1**)

Label symbols:

Represent destinations of goto instructions (**LOOP**, **STOP**, **END**)

Variable symbols:

Represent memory locations where the programmer wants to maintain values

(**i**, **sum**)

Handling Pre-defined Symbols

```
0   @i
1   M=1
2   @sum
3   M=0
(LOOP)
4   @i
5   D=M
6   @R0
7   D=D-M
8   @STOP
9   D;JGT
10  @sum
11  D=M
12  @i
13  D=D+M
14  @sum
15  M=D
16  @i
17  M=M+1
18  @LOOP
19  0;JMP
(STOP)
20  @sum
21  D=M
22  @R1
23  M=D
(END)
24  @END
25  0;JMP
```

Pre-Defined Symbols

The Hack language specification describes 23 pre-defined symbols:

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576

Symbol	Value
SP	0
LCL	1
ARG	2
THIS	3
THAT	4

Translation:

- Predefined symbols occur only in A-instructions, e.g., @predefinedsymbol
- Replace predefinedsymbol with its value

```
0   @i
1   M=1
2   @sum
3   M=0
   (LOOP)
4   @i
5   D=M
6   @R0
7   D=D-M
8   @STOP
9   D;JGT
10  @sum
11  D=M
12  @i
13  D=D+M
14  @sum
15  M=D
16  @i
17  M=M+1
18  @LOOP
19  0;JMP
   (STOP)
20  @sum
21  D=M
22  @R1
23  M=D
   (END)
24  @END
25  0;JMP
```

Handling Labels

Label Symbols

- Used to label destinations of goto commands
- Declared by the pseudo-command (xxx)
- This directive defines the symbol xxx to refer to memory location holding the next instruction in the program

Symbol	Value
LOOP	4
STOP	20
END	24

Translation:

- Label declarations, e.g., (labelsymbol) are not translated, so generate no code and are called pseudo-commands
- Replace labelsymbol with its value, which is the address of the memory location holding the next instruction in the program

```
0      @i
1      M=1
2      @sum
3      M=0
   (LOOP)
4      @i
5      D=M
6      @R0
7      D=D-M
8      @STOP
9      D;JGT
10     @sum
11     D=M
12     @i
13     D=D+M
14     @sum
15     M=D
16     @i
17     M=M+1
18     @LOOP
19     0;JMP
   (STOP)
20     @sum
21     D=M
22     @R1
23     M=D
   (END)
24     @END
25     0;JMP
```

Handling Variables

Variable Symbols

- Any symbol xxx appearing in an assembly program which is not pre- defined and is not defined elsewhere using the `(xxx)` directive is treated as a *variable*
- Each variable is assigned a unique memory address, starting at 16

Symbol	Value
i	16
sum	17

Translation: @varsymbol

- If seen for the first time, assign a unique memory address starting from 16
- Replace varsymbol with the address

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Why Two Pass Assembler?

```
0      @i
1      M=1
2      @sum
3      M=0
   (LOOP)
4      @i
5      D=M
6      @R0
7      D=D-M
8      @STOP
9      D;JGT
10     @sum
11     D=M
12     @i
13     D=D+M
14     @sum
15     M=D
16     @i
17     M=M+1
18     @LOOP
19     0;JMP
   (STOP)
20     @sum
21     D=M
22     @R1
23     M=D
   (END)
24     @END
25     0;JMP
```

First Pass

- Create an empty symbol table

Symbol	Value
...	...

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

First Pass

- Create an empty symbol table
- Initialize the symbol table with the 23 pre-defined symbols

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4

Initialization:

Add the 23 pre-defined symbols

```
0    @i
1    M=1
2    @sum
3    M=0
   (LOOP)
4    @i
5    D=M
6    @R0
7    D=D-M
8    @STOP
9    D;JGT
10   @sum
11   D=M
12   @i
13   D=D+M
14   @sum
15   M=D
16   @i
17   M=M+1
18   @LOOP
19   0;JMP
   (STOP)
20   @sum
21   D=M
22   @R1
23   M=D
   (END)
24   @END
25   0;JMP
```

First Pass

- Create an empty symbol table
- Initialize the symbol table with the 23 pre-defined symbols
- Read the source file and look for label declaration only, and on encountering a label declaration, enter the label name with its corresponding address in the symbol table

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4

Initialization:

Add the 23 pre-defined symbols

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

First Pass

- Create an empty symbol table
- Initialize the symbol table with the 23 pre-defined symbols
- Read the source file and look for label declaration only, and on encountering a label declaration, enter the label name with its corresponding address in the symbol table

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4

Initialization:

Add the 23 pre-defined symbols

First pass: Add the label symbols

```
0    @i
1    M=1
2    @sum
3    M=0
    (LOOP)
4    @i
5    D=M
6    @R0
7    D=D-M
8    @STOP
9    D;JGT
10   @sum
11   D=M
12   @i
13   D=D+M
14   @sum
15   M=D
16   @i
17   M=M+1
18   @LOOP
19   0;JMP
    (STOP)
20   @sum
21   D=M
22   @R1
23   M=D
    (END)
24   @END
25   0;JMP
```

First Pass

- Create an empty symbol table
- Initialize the symbol table with the 23 pre-defined symbols
- Read the source file and look for label declaration only, and on encountering a label declaration, enter the label name with its corresponding address in the symbol table

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20

Initialization:

Add the 23 pre-defined symbols

First pass: Add the label symbols

```
0   @i
1   M=1
2   @sum
3   M=0
   (LOOP)
4   @i
5   D=M
6   @R0
7   D=D-M
8   @STOP
9   D;JGT
10  @sum
11  D=M
12  @i
13  D=D+M
14  @sum
15  M=D
16  @i
17  M=M+1
18  @LOOP
19  0;JMP
   (STOP)
20  @sum
21  D=M
22  @R1
23  M=D
   (END)
24  @END
25  0;JMP
```

First Pass

- Create an empty symbol table
- Initialize the symbol table with the 23 pre-defined symbols
- Read the source file and look for label declaration only, and on encountering a label declaration, enter the label name with its corresponding address in the symbol table

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24

Initialization:

Add the 23 pre-defined symbols

First pass: Add the label symbols

Second Pass

```
0   @i
1   M=1
2   @sum
3   M=0
    (LOOP)
4   @i
5   D=M
6   @R0
7   D=D-M
8   @STOP
9   D;JGT
10  @sum
11  D=M
12  @i
13  D=D+M
14  @sum
15  M=D
16  @i
17  M=M+1
18  @LOOP
19  0;JMP
    (STOP)
20  @sum
21  D=M
22  @R1
23  M=D
    (END)
24  @END
25  0;JMP
```

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16

0	0000000000010000
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16

0	0000000000010000
1	1110111111001000
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16

0	0000000000010000
1	1110111111001000
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	0000000000010000
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	$\boxed{\texttt{D=D+M}}$
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	0000000000010000
13	1111000010010000
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	0000000000010000
13	1111000010010000
14	0000000000010001
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	

Second Pass

0	@i
1	M=1
2	@sum
3	M=0
	(LOOP)
4	@i
5	D=M
6	@R0
7	D=D-M
8	@STOP
9	D;JGT
10	@sum
11	D=M
12	@i
13	D=D+M
14	@sum
15	M=D
16	@i
17	M=M+1
18	@LOOP
19	0;JMP
	(STOP)
20	@sum
21	D=M
22	@R1
23	M=D
	(END)
24	@END
25	0;JMP

Symbol	Value
R0	0
R1	1
R2	2
...	...
R15	15
SCREEN	16384
KBD	24576
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
LOOP	4
STOP	20
END	24
i	16
sum	17

0	0000000000010000
1	1110111111001000
2	0000000000010001
3	1110101010001000
4	0000000000010000
5	1111110000010000
6	0000000000000000
7	1111010011010000
8	0000000000010100
9	1110001100000001
10	0000000000010001
11	1111110000010000
12	0000000000010000
13	1111000010010000
14	0000000000010001
15	1110001100001000
16	0000000000010000
17	1111110111001000
18	0000000000000100
19	1110101010000111
20	0000000000010001
21	1111110000010000
22	0000000000000001
23	1110001100001000
24	0000000000011000
25	1110101010000111

The Two Pass Assembly Process

Initialization:

- Construct an empty symbol table
- Add the pre-defined symbols to the symbol table

First pass:

- Scan the entire program; For each "instruction" of the form (`xxx`):
  - ✓ Add the pair (`xxx, address`) to the symbol table, where address is the number of the instruction following (`xxx`)

Second pass:

- Set n to 16
- Scan the entire program again; For each instruction:
  - ✓ If the instruction is `@symbol`, look up symbol in the symbol table;
    - o If (`symbol, value`) is found, use value to complete the instruction's translation;
    - o If NOT found:
      - ❖ Add (`symbol, n`) to the symbol table;
      - ❖ Use n to complete the instruction's translation;
      - ❖ Do n++
  - ✓ If the instruction is a C-instruction, complete the instruction's translation
  - ✓ Write the translated instruction to the output file

Hack Assembler Implementation C/C++/Python

Things To Do


O.k., and now you'll do exactly what I'm telling you !
Access denied