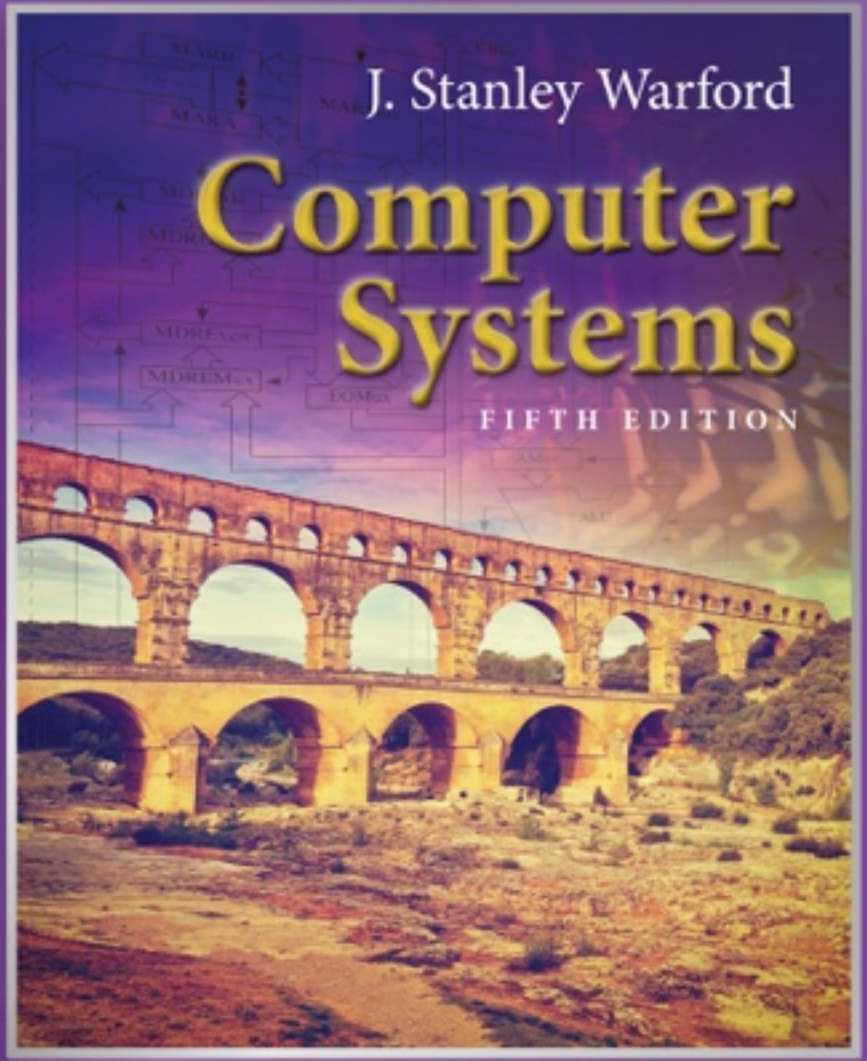
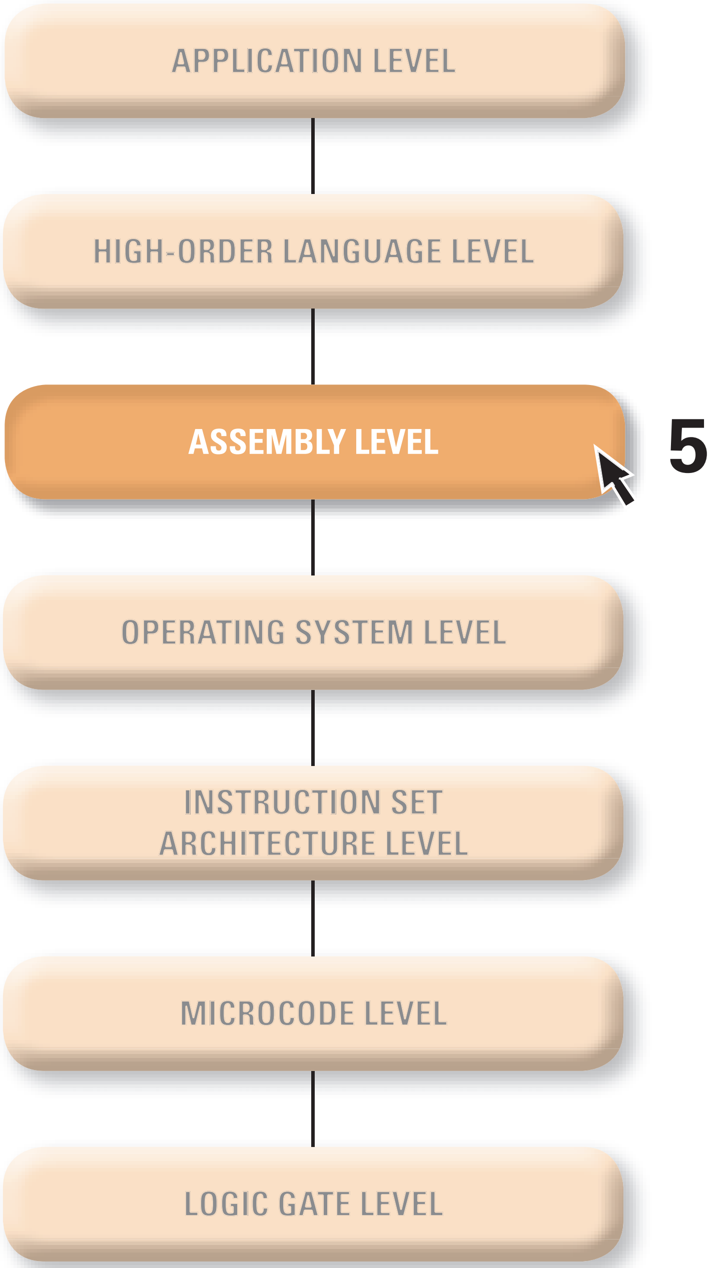
Chapter 5

Assembly Language



Assembly



APPLICATION LEVEL

HIGH-ORDER LANGUAGE LEVEL

ASSEMBLY LEVEL

5

OPERATING SYSTEM LEVEL

INSTRUCTION SET ARCHITECTURE LEVEL

MICROCODE LEVEL

LOGIC GATE LEVEL

Two types of bit patterns

- Instructions
  - Mnemonics for opcodes
  - Letters for addressing modes
- Data
  - Pseudo-ops, also called dot commands

Figure 5.1

aaa	Addressing Mode	Letters
000	Immediate	i
001	Direct	d
010	Indirect	n
011	Stack-relative	s
100	Stack-relative deferred	sf
101	Indexed	x
110	Stack-indexed	sx
111	Stack-deferred indexed	sfx

Figure 5.2

Instruction Specifier	Mnemonic	Instruction	Addressing Mode	Status Bits
0000 0000	STOP	Stop execution	U	
0000 0001	RET	Return from CALL	U	
0000 0010	RETTR	Return from trap	U	
0000 0011	MOVSPA	Move SP to A	U	
0000 0100	MOVFLGA	Move NZVC flags to A⟨12..15⟩	U	
0000 0101	MOVAFLG	Move A⟨12..15⟩ to NZVC flags	U	
0000 011r	NOTr	Bitwise invert r	U	NZ
0000 100r	NEGr	Negate r	U	NZV
0000 101r	ASLr	Arithmetic shift left r	U	NZVC
0000 110r	ASRr	Arithmetic shift right r	U	NZC
0000 111r	ROLr	Rotate left r	U	C
0001 000r	RORr	Rotate right r	U	C

Figure 5.2 (continued)

0001 001a	BR	Branch unconditional	i, x
0001 010a	BRLE	Branch if less than or equal to	i, x
0001 011a	BRLT	Branch if less than	i, x
0001 100a	BREQ	Branch if equal to	i, x
0001 101a	BRNE	Branch if not equal to	i, x
0001 110a	BRGE	Branch if greater than or equal to	i, x
0001 111a	BRGT	Branch if greater than	i, x
0010 000a	BRV	Branch if V	i, x
0010 001a	BRC	Branch if C	i, x
0010 010a	CALL	Call subroutine	i, x
0010 011n	NOPn	Unary no operation trap	U
0010 1aaa	NOP	Nonunary no operation trap	i

Figure 5.2 (continued)

0011 0aaa	DECI	Decimal input trap	d, n, s, sf, x, sx, sfx	NZV
0011 1aaa	DECO	Decimal output trap	i, d, n, s, sf, x, sx, sfx	
0100 0aaa	HEXO	Hexadecimal output trap	i, d, n, s, sf, x, sx, sfx	
0100 1aaa	STRO	String output trap	d, n, s, sf, x	
0101 0aaa	ADDSP	Add to stack pointer (SP)	i, d, n, s, sf, x, sx, sfx	NZVC
0101 1aaa	SUBSP	Subtract from stack pointer (SP)	i, d, n, s, sf, x, sx, sfx	NZVC

Figure 5.2 (continued)

0110 raaa	`ADDr`	Add to r	i, d, n, s, sf, x, sx, sfx	NZVC
0111 raaa	`SUBr`	Subtract from r	i, d, n, s, sf, x, sx, sfx	NZVC
1000 raaa	`ANDr`	Bitwise AND to r	i, d, n, s, sf, x, sx, sfx	NZ
1001 raaa	`ORr`	Bitwise OR to r	i, d, n, s, sf, x, sx, sfx	NZ
1010 raaa	`CPWr`	Compare word to r	i, d, n, s, sf, x, sx, sfx	NZVC
1011 raaa	`CPBr`	Compare byte to r⟨8..15⟩	i, d, n, s, sf, x, sx, sfx	NZVC
1100 raaa	`LDWr`	Load word r from memory	i, d, n, s, sf, x, sx, sfx	NZ
1101 raaa	`LDBr`	Load byte r⟨8..15⟩ from memory	i, d, n, s, sf, x, sx, sfx	NZ
1110 raaa	`STWr`	Store word r to memory	d, n, s, sf, x, sx, sfx	
1111 raaa	`STBr`	Store byte r⟨8..15⟩ to memory	d, n, s, sf, x, sx, sfx	

The unimplemented opcode instructions

• `NOPn`	Unary no-operation trap
• `NOP`	Nonunary no-operation trap
• `DECI`	Decimal input trap
• `DECO`	Decimal output trap
• `HEXO`	Hexadecimal output trap
• `STRO`	String output trap

Pseudo-operations

`.ADDRSS`	The address of a symbol
`.ALIGN`	Padding to align at a memory boundary
`.ASCII`	A string of ASCII bytes
`.BLOCK`	A block of zero bytes
`.BURN`	Initiate ROM burn
`.BYTE`	A byte value
`.END`	The sentinel for the assembler
`.EQUATE`	Equate a symbol to a constant value
`.WORD`	A word value

Figure 5.3

Assembler Input

```
;Stan Warford
;May 1, 2017
;A program to output "Hi"
;
LDBA    0x000D,d    ;Load byte accumulator 'H'
STBA    0xFC16,d    ;Store byte accumulator output device
LDBA    0x000E,d    ;Load byte accumulator 'i'
STBA    0xFC16,d    ;Store byte accumulator output device
STOP                ;Stop
.ASCII  "Hi"        ;ASCII "Hi" characters
.END
```

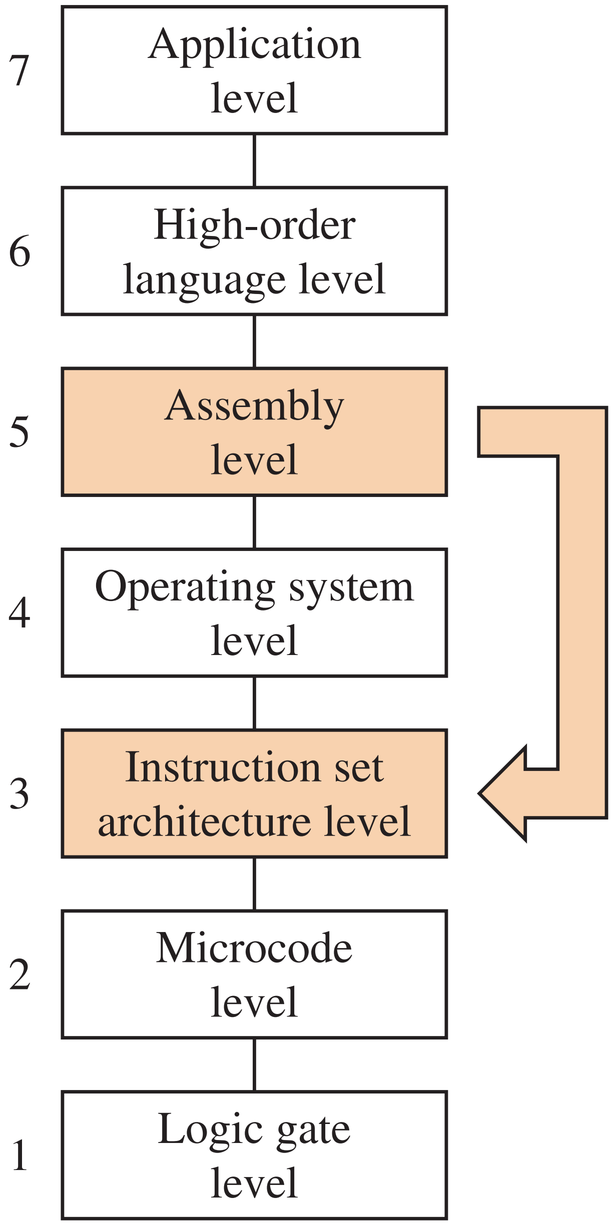
Assembler Output

```
D1 00 0D F1 FC 16 D1 00 0E F1 FC 16 00 48 69 zz
```

Program Output

```
Hi
```

Figure 5.4



7 Application level

6 High-order language level

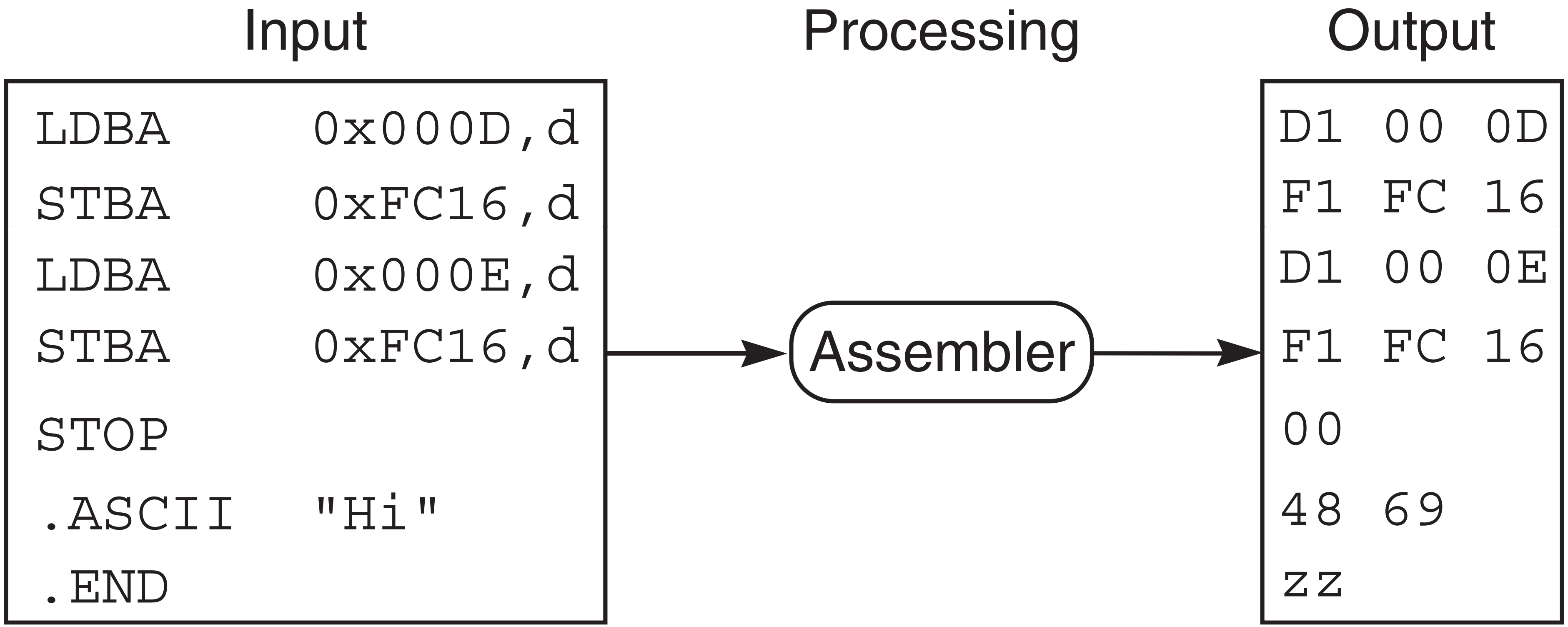
5 Assembly level

4 Operating system level

3 Instruction set architecture level

2 Microcode level

1 Logic gate level



```
LDBA      0x000D,d
STBA      0xFC16,d
LDBA      0x000E,d
STBA      0xFC16,d
STOP
.ASCII    "Hi"
.END
```

```
D1 00 0D
F1 FC 16
D1 00 0E
F1 FC 16
00
48 69
zz
```

Figure 5.6

Assembler Input

```
LDBA     0xFC15,d     ;Input first character
STBA     0x0013,d     ;Store first character
LDBA     0xFC15,d     ;Input second character
STBA     0xFC16,d     ;Output second character
LDBA     0x0013,d     ;Load first character
STBA     0xFC16,d     ;Output first character
STOP                  ;Stop
.BLOCK   1            ;Storage for first character
.END
```

Assembler Output

```
D1 FC 15 F1 00 13 D1 FC 15 F1 FC 16 D1 00 13 F1
FC 16 00 00 zz
```

Program Input

```
up
```

Program Output

```
pu
```

Figure 5.7

Assembler Input

```
LDWA     0x000D,d     ;A <- first number
ADDA     0x000F,d     ;Add the two numbers
ORA      0x0011,d     ;Convert sum to character
STBA     0xFC16,d     ;Output the character
STOP                  ;Stop
.WORD    5            ;Decimal 5
.WORD    3            ;Decimal 3
.WORD    0x0030       ;Mask for ASCII char
.END
```

Assembler Output

```
C1 00 0D 61 00 0F 91 00 11 F1 FC 16 00 00 05 00
03 00 30 zz
```

Program Output

```
8
```

Computer Systems FIFTH EDITION

Figure 5.8

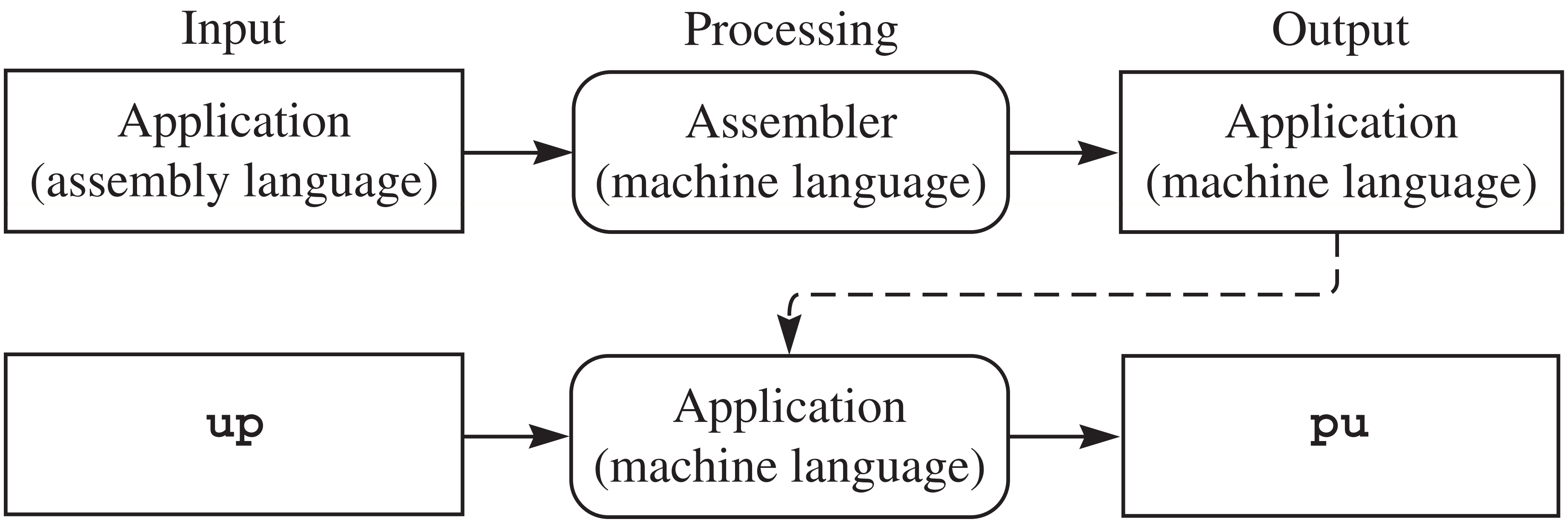


Figure 5.9

Assembler Input

```
         ldwa 0x000D,d    ;A <- first number
 ADda    0x000F,d ;Add the two numbers
         ORA     0x0011, d    ;Convert sum to character
   StBA    0Xfc16    ,      d    ;Output the character
         STop    ;Stop
  .WORD 5           ;Decimal 5
 .worD     3   ;Decimal 3
      .WORD   0x0030  ;Mask for ASCII char
    .end
```

Assembler Listing

```
-----------------------------------------------------------------
      Object
Addr  code      Mnemon  Operand     Comment
-----------------------------------------------------------------
0000  C1000D    LDWA    0x000D,d    ;A <- first number
0003  61000F    ADDA    0x000F,d    ;Add the two numbers
0006  910011    ORA     0x0011,d    ;Convert sum to character
0009  F1FC16    STBA    0xFC16,d    ;Output the character
000C  00        STOP                ;Stop
000D  0005      .WORD   5           ;Decimal 5
000F  0003      .WORD   3           ;Decimal 3
0011  0030      .WORD   0x0030      ;Mask for ASCII char
0013            .END
-----------------------------------------------------------------
```

Direct addressing

- Oprnd = Mem[OprndSpec]
- Asmb5 letter: `d`
- The operand specifier is the *address* in memory of the operand.

Immediate addressing

- Oprnd = OprndSpec
- Asmb5 letter: `i`
- The operand specifier *is* the operand.

Figure 5.10

```
LDBA    'H',i         ;Output 'H'
STBA    0xFC16,d
LDBA    'i',i         ;Output 'i'
STBA    0xFC16,d
STOP
.END
```

Output

```
Hi
```

The decimal input instruction

- Instruction specifier: 0011 0aaa
- Mnemonic: `DECI`
- Convert a string of ASCII characters from the input device into a 16-bit signed integer and store it into memory

$$\text{Oprnd} \leftarrow \{\textit{decimal input}\}$$

The decimal output instruction

- Instruction specifier: 0011 1aaa
- Mnemonic: `DECO`
- Convert a 16-bit signed integer from memory into a string of ASCII characters and send the string to the output device

$$\{\mathit{decimal\ output}\} \leftarrow \text{Oprnd}$$

The unconditional branch instruction

- Instruction specifier: 0001 001a
- Mnemonic: `BR`
- Skips to a different memory location for the next instruction to be executed.

$$\mathrm{PC} \leftarrow \{\mathrm{Oprnd}\}$$

Figure 5.11

```
0000  120005    BR      0x0005      ;Branch around data
0003  0000      .BLOCK  2           ;Storage for one integer
                ;
0005  310003    DECI    0x0003,d    ;Get the number
0008  390003    DECO    0x0003,d    ;and output it
000B  D00020    LDBA    ' ',i       ;Output " + 1 = "
000E  F1FC16    STBA    0xFC16,d
0011  D0002B    LDBA    '+',i
0014  F1FC16    STBA    0xFC16,d
0017  D00020    LDBA    ' ',i
001A  F1FC16    STBA    0xFC16,d
001D  D00031    LDBA    '1',i
0020  F1FC16    STBA    0xFC16,d
0023  D00020    LDBA    ' ',i
0026  F1FC16    STBA    0xFC16,d
0029  D0003D    LDBA    '=',i
002C  F1FC16    STBA    0xFC16,d
002F  D00020    LDBA    ' ',i
0032  F1FC16    STBA    0xFC16,d
0035  C10003    LDWA    0x0003,d    ;A <- the number
0038  600001    ADDA    1,i         ;Add one to it
003B  E10003    STWA    0x0003,d    ;Store the sum
003E  390003    DECO    0x0003,d    ;Output the sum
0041  00        STOP
0042            .END
```

Figure 5.11 (continued)

Input

```
-479
```

Output

```
-479 + 1 = -478
```

The string output instruction

- Instruction specifier: 0100 1aaa
- Mnemonic: `STRO`
- Send a string of null-terminated ASCII characters to the output device

$$\{\mathit{string\ output}\} \leftarrow \mathrm{Oprnd}$$

Figure 5.12

```
0000  120005    BR      0x0005      ;Branch around data
0003  0000      .BLOCK  2           ;Storage for one integer
                ;
0005  310003    DECI    0x0003,d    ;Get the number
0008  390003    DECO    0x0003,d    ;and output it
000B  49001B    STRO    0x001B,d    ;Output " + 1 = "
000E  C10003    LDWA    0x0003,d    ;A <- the number
0011  600001    ADDA    1,i         ;Add one to it
0014  E10003    STWA    0x0003,d    ;Store the sum
0017  390003    DECO    0x0003,d    ;Output the sum
001A  00        STOP
001B  202B20    .ASCII  " + 1 = \x00"
      31203D
      2000
0023            .END
```

Input

```
-479
```

Output

```
-479 + 1 = -478
```

The hexadecimal output instruction

- Instruction specifier: 0100 0aaa
- Mnemonic: `HEXO`
- Convert a 2-byte word from memory into four hexadecimal digits and send the string to the output device

$$\{\textit{hexadecimal output}\} \leftarrow \mathrm{Oprnd}$$

Interpreting bit patterns

- Dot commands set bit patterns at assembly time
- Executable statements interpret bit patterns at run time

```
0000  120009   BR     0x0009     ;Branch around data
0003  FFFE     .WORD  0xFFFE     ;First
0005  00       .BYTE  0x00       ;Second
0006  55       .BYTE  'U'        ;Third
0007  0470     .WORD  1136       ;Fourth
               ;
0009  390003   DECO   0x0003,d   ;Interpret First as dec
000C  D0000A   LDBA   '\n',i
000F  F1FC16   STBA   0xFC16,d
0012  390005   DECO   0x0005,d   ;Interpret Second and Third as dec
0015  F1FC16   STBA   0xFC16,d
0018  D0000A   LDBA   '\n',i
001B  410005   HEXO   0x0005,d   ;Interpret Second and Third as hex
001E  D0000A   LDBA   '\n',i
0021  F1FC16   STBA   0xFC16,d
0024  D10006   LDBA   0x0006,d   ;Interpret Third as char
0027  F1FC16   STBA   0xFC16,d
002A  D10008   LDBA   0x0008,d   ;Interpret Fourth as char
002D  F1FC16   STBA   0xFC16,d
0030  00       STOP
0031           .END
```

Figure 5.13 (continued)

Output

```
-2
85
0055
Up
```

Disassembler

- The inverse mapping of an assembler is not unique
- Given a bit pattern at level ISA3, you cannot determine the Asmb5 statement that produced it

Assembly Language Program

```
0000   D10013     LDBA     0x0013,d
0003   F1FC16     STBA     0xFC16,d
0006   D10014     LDBA     0x0014,d
0009   F1FC16     STBA     0xFC16,d
000C   D10015     LDBA     0x0015,d
000F   F1FC16     STBA     0xFC16,d
0012   00         STOP
0013   50756E     .ASCII   "Pun"
0016              .END
```

Assembly Language Program

```
0000   D10013     LDBA     0x0013,d
0003   F1FC16     STBA     0xFC16,d
0006   D10014     LDBA     0x0014,d
0009   F1FC16     STBA     0xFC16,d
000C   D10015     LDBA     0x0015,d
000F   F1FC16     STBA     0xFC16,d
0012   00         STOP
0013   50756E     ADDSP    0x756E,i
0016              .END
```

Program Output

```
Pun
```

Mappings

- The mapping from Asmb5 to ISA3 is one-to-*one*
- The mapping from HOL6 to Asmb5 is one-to-*many*

Symbols

- Defined by an identifier followed by a colon at the start of a statement
- The value of a symbol is the address of the object code generated by the statement

Figure 5.15

Assembler Listing

```
------------------------------------------------------------------------
      Object
Addr  code   Symbol   Mnemon  Operand     Comment
------------------------------------------------------------------------
0000  120005          BR      main        ;Branch around data
0003  0000   num:     .BLOCK  2           ;Storage for one integer #2d
             ;
0005  310003 main:    DECI    num,d       ;Get the number
0008  390003          DECO    num,d       ;and output it
000B  49001B          STRO    msg,d       ;Output " + 1 = "
000E  C10003          LDWA    num,d       ;A <- the number
0011  600001          ADDA    1,i         ;Add one to it
0014  E10003          STWA    num,d       ;Store the sum
0017  390003          DECO    num,d       ;Output the sum
001A  00              STOP
001B  202B20 msg:     .ASCII  " + 1 = \x00"
      31203D
      2000
0023                  .END
------------------------------------------------------------------------
```

Figure 5.15 (continued)

```
Symbol table
--------------------------------------
Symbol     Value        Symbol     Value
--------------------------------------
main       0005         msg        001B
num        0003
--------------------------------------
```

Input

```
-479
```

Output

```
-479 + 1 = -478
```

Assembler Input

```
this:    DECO    this,d
         STOP
         .END
```

Assembler Listing

```
0000  390000 this:    DECO    this,d
0003  00              STOP
0004                  .END
```

Output

```
14592
```

Computer Systems FIFTH EDITION

Figure 5.17

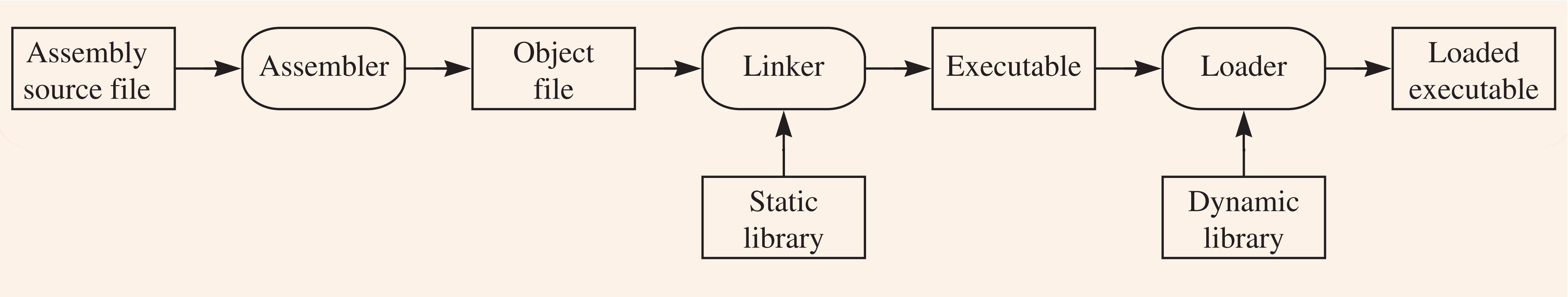
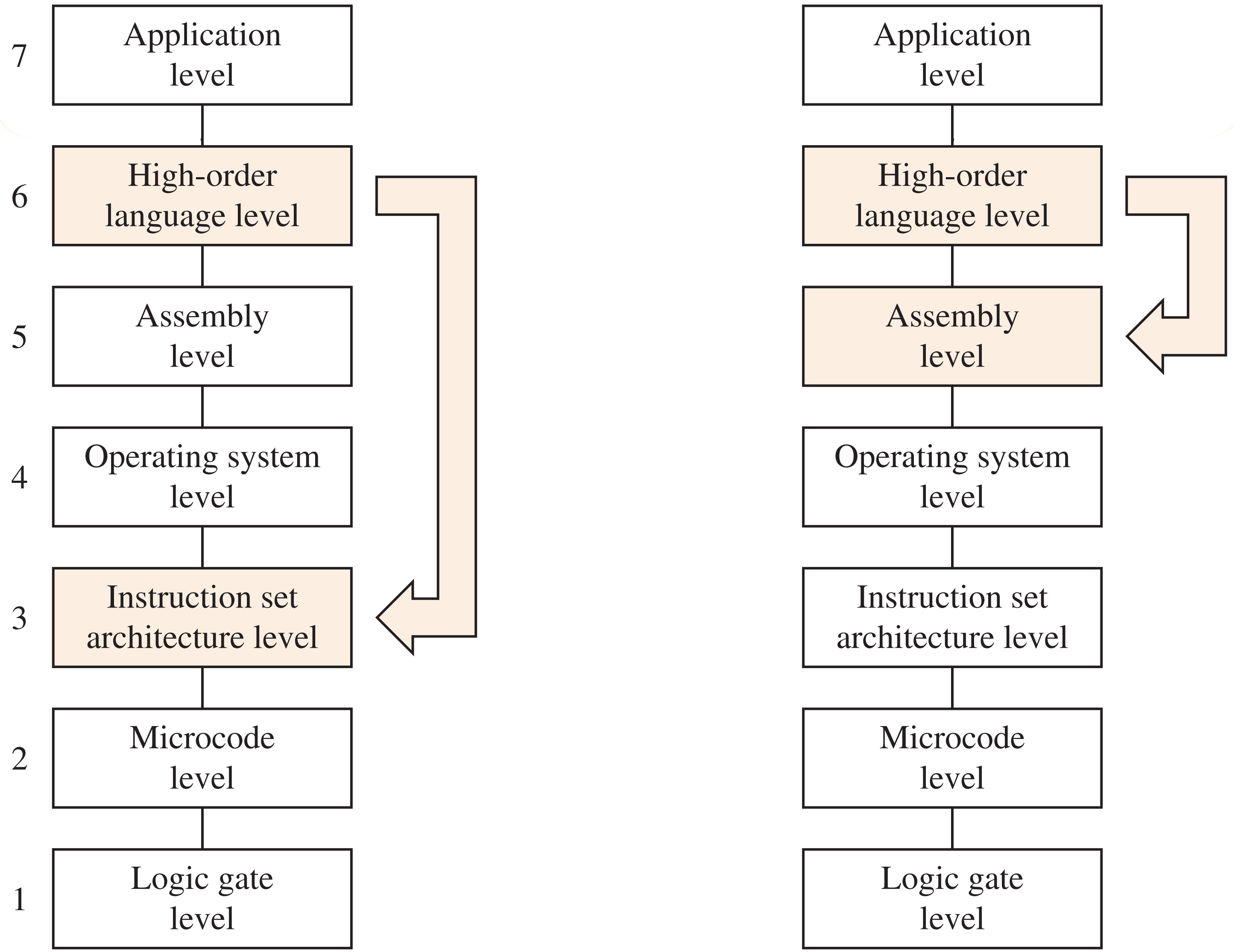


Figure 5.18



(a) Translation directly to machine language.

(b) Translation to assembly language.

Translating `printf()`

- Translate string output with `STRO`
- Translate character output with

  `STBA charOut,d`
- Translate integer output with `DECO`

Figure 5.19

High-Order Language

```
#include <stdio.h>
int main() {
   printf("Hello, world!\n");
   return 0;
}
```

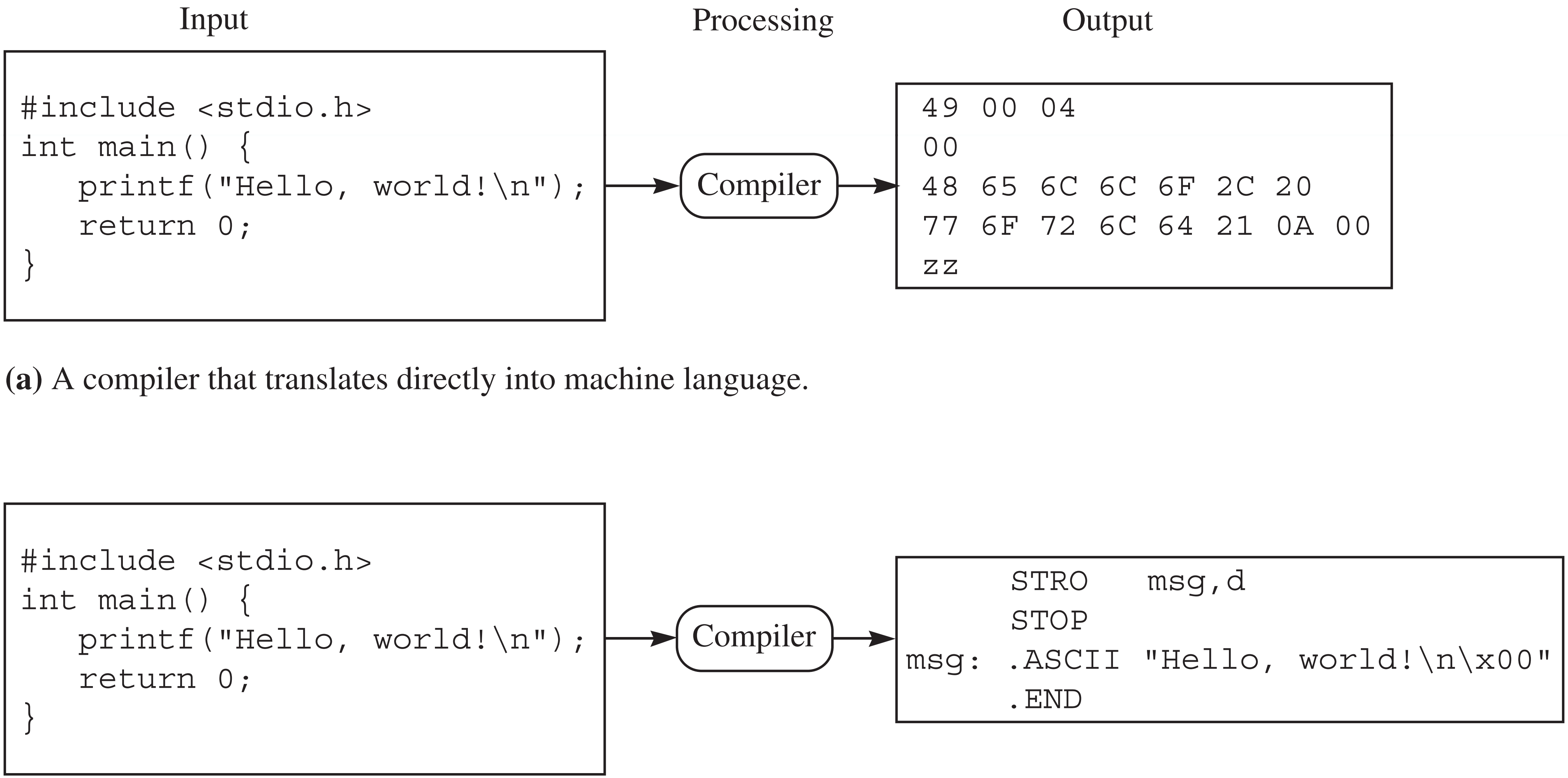
Assembly Language

```
0000  490004                STRO    msg,d
0003  00                    STOP
0004  48656C msg:           .ASCII  "Hello, world!\n\x00"
      6C6F2C
      20776F
      726C64
      210A00
0013                        .END
```

Output

```
Hello, world!
```

Figure 5.20



```
#include <stdio.h>
int main() {
   printf("Hello, world!\n");
   return 0;
}
```

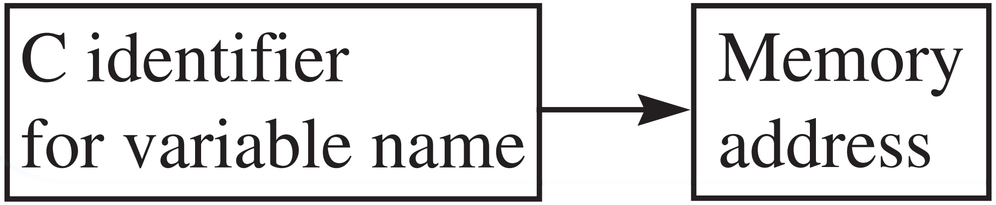
```
49 00 04
00
48 65 6C 6C 6F 2C 20
77 6F 72 6C 64 21 0A 00
zz
```

(a) A compiler that translates directly into machine language.

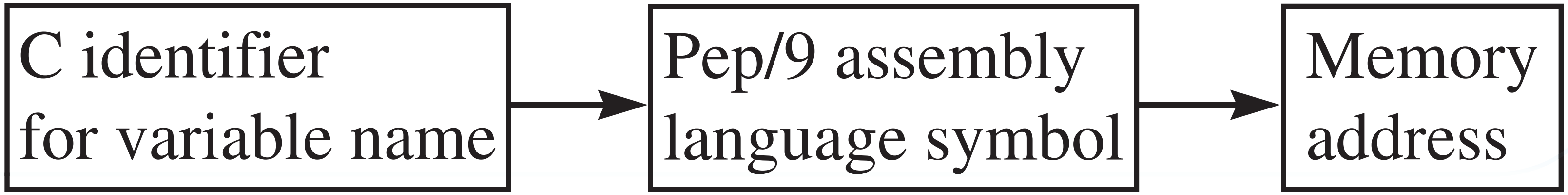
```
#include <stdio.h>
int main() {
   printf("Hello, world!\n");
   return 0;
}
```

```
        STRO    msg,d
        STOP
msg:    .ASCII  "Hello, world!\n\x00"
        .END
```

(b) A compiler that translates into assembly language.



(a) A compiler that translates to machine language.



(b) A hypothetical compiler for illustrative purposes.

Global variables

- Allocated at a fixed location in memory with `.BLOCK`
- Accessed with direct addressing (`d`)

Assignment statements

- Load the accumulator from the right hand side of the assignment with `LDWA` or `LDBA`
- Compute the value of the right hand side of the assignment if necessary
- Store the value to the variable on the left hand side of the assignment with `STWA` or `STBA`

Translating `scanf()`

- Translate character input with

```
LDBA charIn,d
```

- Translate integer input with `DECI`

High-Order Language

```
#include <stdio.h>
char ch;
int j;
int main() {
   scanf("%c %d", &ch, &j);
   j += 5;
   ch++;
   printf("%c\n%d\n", ch, j);
   return 0;
}
```

Figure 5.22 (continued)

Assembly Language

```
0000  120006         BR      main
0003  00      ch:    .BLOCK  1           ;global variable #1c
0004  0000    j:     .BLOCK  2           ;global variable #2d
              ;
0006  D1FC15  main:  LDBA    charIn,d    ;scanf("%c %d", &ch, &j)
0009  F10003         STBA    ch,d
000C  310004         DECI    j,d
000F  C10004         LDWA    j,d         ;j += 5
0012  600005         ADDA    5,i
0015  E10004         STWA    j,d
0018  D10003         LDBA    ch,d        ;ch++
001B  600001         ADDA    1,i
001E  F10003         STBA    ch,d
0021  D10003         LDBA    ch,d        ;printf("%c\n%d\n", ch, j)
0024  F1FC16         STBA    charOut,d
0027  D0000A         LDBA    '\n',i
002A  F1FC16         STBA    charOut,d
002D  390004         DECO    j,d
0030  D0000A         LDBA    '\n',i
0033  F1FC16         STBA    charOut,d
0036  00             STOP
0037                 .END
```

Figure 5.22 (continued)

Input

```
M 419
```

Output

```
N
424
```

Figure 5.23

```
#include <stdio.h>
char ch;
int j;
int main() {
   scanf("%c %d", &ch, &j);
   j += 5;
   ch++;
   printf("%c\n%d\n", ch, j);
   return 0;
}
```

	symbol	value	kind
[0]	ch	0003	sChar
[1]	j	0004	sInt
[2]	⋮	⋮	⋮

Figure 5.1

```
#include <stdio.h>

int j;
float y;

int main () {
   ...
   j = j % 8;
   ...
   y = y % 8; // Compile error
   ...
}
```

	symbol	value	kind
[0]	j	0003	sInt
[1]	y	0005	sFloat
[2]	⋮	⋮	⋮

Trace tags

- Format trace tags
  - Required for global and local variables
- Symbol trace tags
  - Not required for global variables

Format trace tags

`#1c`	One-byte character
`#1d`	One-byte decimal
`#2d`	Two-byte decimal
`#1h`	One-byte hexadecimal
`#2h`	Two-byte hexadecimal

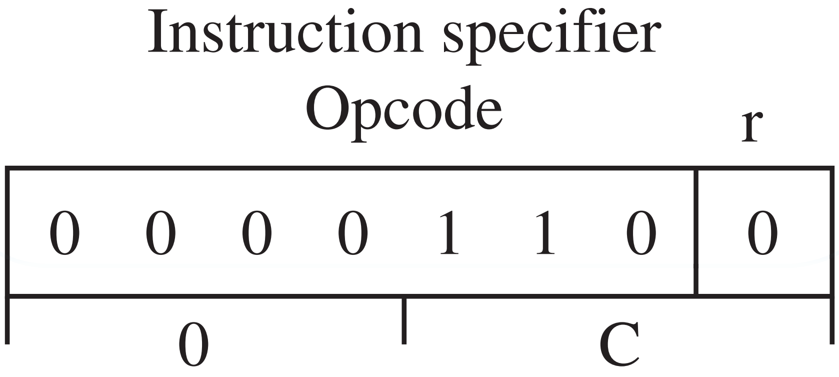
The arithmetic shift right instruction

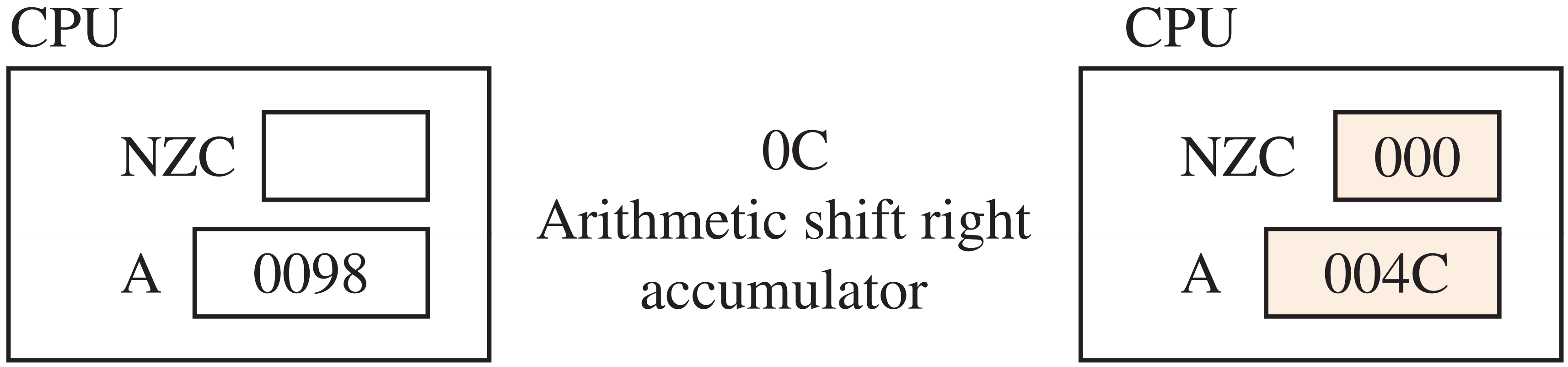
- Instruction specifier: 0000 110r
- Mnemonic: ASRr (ASRA, ASRX)
- Performs a one-bit arithmetic shift right on a 16-bit register

$$C \leftarrow r\langle 15\rangle \;,\; r\langle 1..15\rangle \leftarrow r\langle 0..14\rangle;$$

$$N \leftarrow r < 0 \;,\; Z \leftarrow r = 0$$

Figure 5.25, 5.26





(a) Before.

(b) After.

The arithmetic shift left instruction

- Instruction specifier: 0000 101r
- Mnemonic: `ASLr` (`ASLA`, `ASLX`)
- Performs a one-bit arithmetic shift left on a 16-bit register

$$C \leftarrow r\langle 0 \rangle \;,\; r\langle 0..14 \rangle \leftarrow r\langle 1..15 \rangle \;,\; r\langle 15 \rangle \leftarrow 0 \;;$$

$$N \leftarrow r < 0 \;,\; Z \leftarrow r = 0 \;,\; V \leftarrow \{\mathit{overflow}\}$$

The rotate left instruction

- Instruction specifier: 0000 111r
- Mnemonic: `ROLr` (`ROLA`, `ROLX`)
- Performs a one-bit rotate left on a 16-bit register

$$C \leftarrow r\langle 0 \rangle \;,\; r\langle 0..14 \rangle \leftarrow r\langle 1..15 \rangle \;,\; r\langle 15 \rangle \leftarrow C \;;$$

The rotate right instruction

- Instruction specifier: 0001 000r
- Mnemonic: `RORr` (`RORA`, `RORX`)
- Performs a one-bit rotate right on a 16-bit register

$$C \leftarrow r\langle 15\rangle \text{ , } r\langle 1..15\rangle \leftarrow r\langle 0..14\rangle \text{ , } r\langle 0\rangle \leftarrow C \text{ ;}$$

Constants

- Equate the constant to its value with `.EQUATE`
- `.EQUATE` does not generate object code
- The value of the constant symbol is not an address

High-Order Language

```
#include <stdio.h>

const int bonus = 10;
int exam1;
int exam2;
int score;


int main() {
   scanf("%d %d", &exam1, &exam2);
   score = (exam1 + exam2) / 2 + bonus;
   printf("score = %d\n", score);
   return 0;
}
```

Figure 5.27 (continued)

Assembly Language

```
0000   120009          BR      main
              bonus:   .EQUATE 10          ;constant
0003   0000   exam1:   .BLOCK  2           ;global variable #2d
0005   0000   exam2:   .BLOCK  2           ;global variable #2d
0007   0000   score:   .BLOCK  2           ;global variable #2d
              ;
0009   310003 main:    DECI    exam1,d     ;scanf("%d %d", &exam1, &exam2)
000C   310005          DECI    exam2,d
000F   C10003          LDWA    exam1,d     ;score = (exam1 + exam2) / 2 + bonus
0012   610005          ADDA    exam2,d
0015   0C              ASRA
0016   60000A          ADDA    bonus,i
0019   E10007          STWA    score,d
001C   490029          STRO    msg,d       ;printf("score = %d\n", score)
001F   390007          DECO    score,d
0022   D0000A          LDBA    '\n',i
0025   F1FC16          STBA    charOut,d
0028   00              STOP
0029   73636F msg:     .ASCII  "score = \x00"
       726520
       3D2000
0032                   .END
```

Figure 5.27 (continued)

```
Symbol table
--------------------------------------
Symbol     Value        Symbol     Value
--------------------------------------
bonus      000A         exam1      0003
exam2      0005         main       0009
msg        0029         score      0007
--------------------------------------
```

Input

```
68 84
```

Output

```
score = 86
```

Figure 5.28

Assembly Language

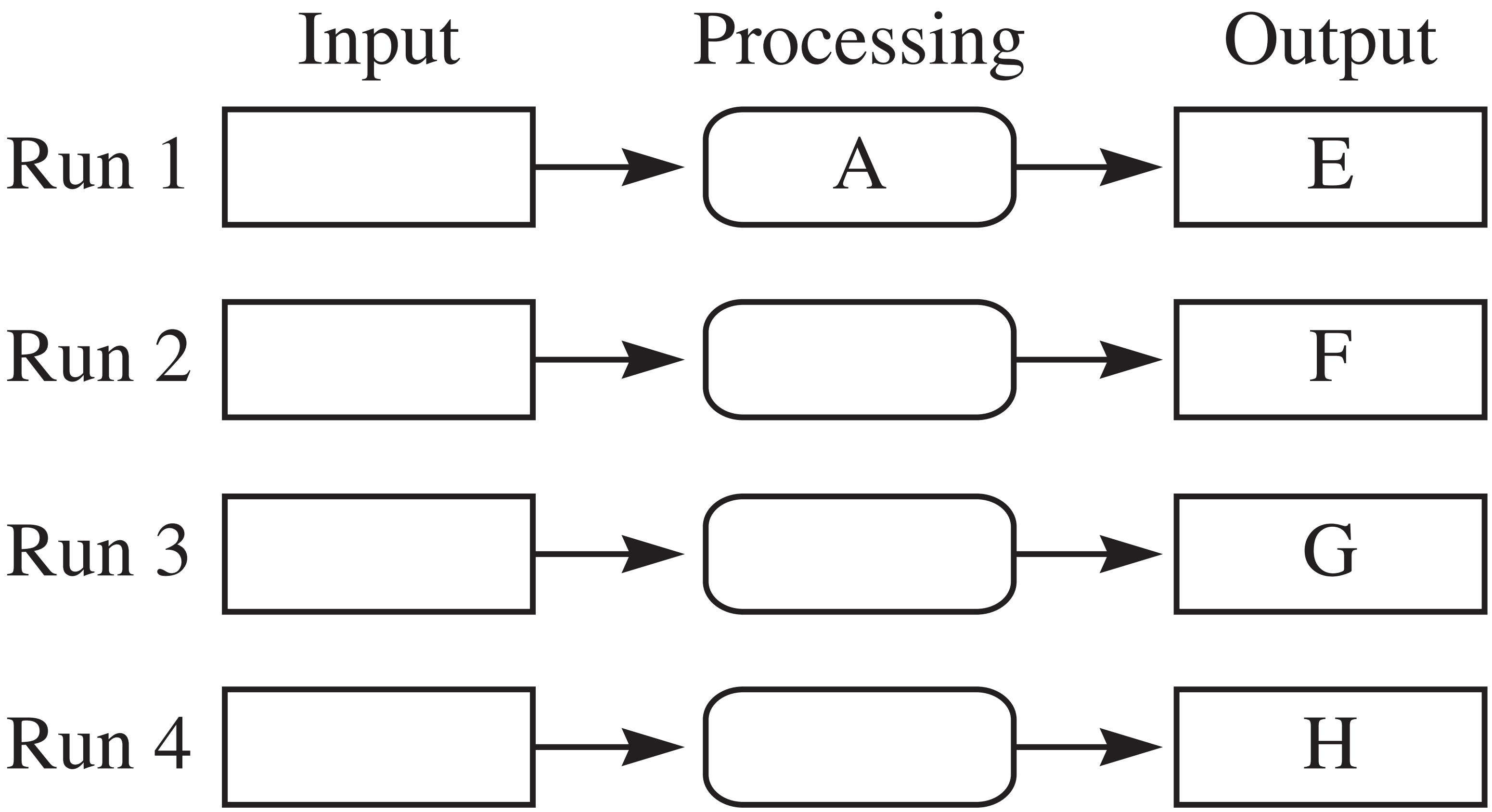
```
0000  310020 main:    DECI    exam1,d     ;scanf("%d %d", &exam1,
0003  310022          DECI    exam2,d     ; &exam2)
0006  C10020          LDWA    exam1,d     ;score = (exam1
0009  610022          ADDA    exam2,d     ; + exam2)
000C  0C              ASRA                ; / 2
000D  60000A          ADDA    bonus,i     ; + bonus
0010  E10024          STWA    score,d
0013  490026          STRO    msg,d       ;printf("score = %d\n",
0016  390024          DECO    score,d     ; score)
0019  D0000A          LDBA    '\n',i
001C  F1FC16          STBA    charOut,d
001F  00              STOP
             ;
             bonus:   .EQUATE 10          ;constant
0020  0000   exam1:   .BLOCK  2           ;global variable #2d
0022  0000   exam2:   .BLOCK  2           ;global variable #2d
0024  0000   score:   .BLOCK  2           ;global variable #2d
0026  73636F msg:     .ASCII  "score = \x00"
      726520
      3D2000
002F                  .END
```

Figure 5.28 (continued)

Assembly Language

```
Symbol table
--------------------------------------
Symbol    Value        Symbol    Value
--------------------------------------
bonus     000A         exam1     0020
exam2     0022         main      0000
msg       0026         score     0024
--------------------------------------
```

Figure 5.29


Input
Processing
Output
Run 1
A
E
Run 2
F
Run 3
G
Run 4
H