

Assembly Language

Two types of bit patterns

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

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-indexed deferred	sxf

Instruction Specifier	Mnemonic	Instruction	Addressing Modes	Status Bits
0000 0000	STOP	Stop execution	U	
0000 0001	RETR	Return from trap	U	
0000 0010	MOVSPA	Move SP to A	U	
0000 0011	MOVFLGA	Move NZVC flags to A	U	
0000 010a	BR	Branch unconditional	i, x	
0000 011a	BRLE	Branch if less than or equal to	i, x	
0000 100a	BRLT	Branch if less than	i, x	
0000 101a	BREQ	Branch if equal to	i, x	
0000 110a	BRNE	Branch if not equal to	i, x	
0000 111a	BRGE	Branch if greater than or equal to	i, x	
0001 000a	BRGT	Branch if greater than	i, x	
0001 001a	BRV	Branch if V	i, x	
0001 010a	BRC	Branch if C	i, x	
0001 011a	CALL	Call subroutine	i, x	

0001 100r	NOTr	Bitwise invert r	U	NZ
0001 101r	NEGr	Negate r	U	NZV
0001 110r	ASLr	Arithmetic shift left r	U	NZVC
0001 111r	ASRr	Arithmetic shift right r	U	NZC
0010 000r	ROLr	Rotate left r	U	C
0010 001r	RORr	Rotate right r	U	C
0010 01nn	NOPn	Unary no operation trap	U	
0010 1aaa	NOP	Nonunary no operation trap	i	
0011 0aaa	DECI	Decimal input trap	d, n, s, sf, x, sx, sxf	NZV
0011 1aaa	DECO	Decimal output trap	i, d, n, s, sf, x, sx, sxf	
0100 0aaa	STRO	String output trap	d, n, sf	
0100 1aaa	CHARI	Character input	d, n, s, sf, x, sx, sxf	
0101 0aaa	CHARO	Character output	i, d, n, s, sf, x, sx, sxf	
0101 1nnn	RETn	Return from call with n local bytes	U	

0110 0aaa	ADDSP	Add to stack pointer (SP)	i, d, n, s, sf, x, sx, sxf	NZVC
0110 1aaa	SUBSP	Subtract from stack pointer (SP)	i, d, n, s, sf, x, sx, sxf	NZVC
0111 raaa	ADD _r	Add to r	i, d, n, s, sf, x, sx, sxf	NZVC
1000 raaa	SUB _r	Subtract from r	i, d, n, s, sf, x, sx, sxf	NZVC
1001 raaa	AND _r	Bitwise AND to r	i, d, n, s, sf, x, sx, sxf	NZ
1010 raaa	OR _r	Bitwise OR to r	i, d, n, s, sf, x, sx, sxf	NZ
1011 raaa	CPr	Compare r	i, d, n, s, sf, x, sx, sxf	NZVC
1100 raaa	LDr	Load r from memory	i, d, n, s, sf, x, sx, sxf	NZ
1101 raaa	LDBYTE _r	Load byte from memory	i, d, n, s, sf, x, sx, sxf	NZ
1110 raaa	ST _r	Store r to memory	d, n, s, sf, x, sx, sxf	
1111 raaa	STBYTE _r	Store byte r to memory	d, n, s, sf, x, sx, sxf	

The unimplemented opcode instructions

- **NOPn** Unary no operation trap
- **NOP** Nonunary no operation trap
- **DECI** Decimal input trap
- **DECO** Decimal output trap
- **STRO** String output trap

Pseudo-operations

- `.ADDRSS` The address of a symbol
- `.ASCII` A string of ASCII bytes
- `.BLOCK` A block of 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 (two bytes)

Assembler Input

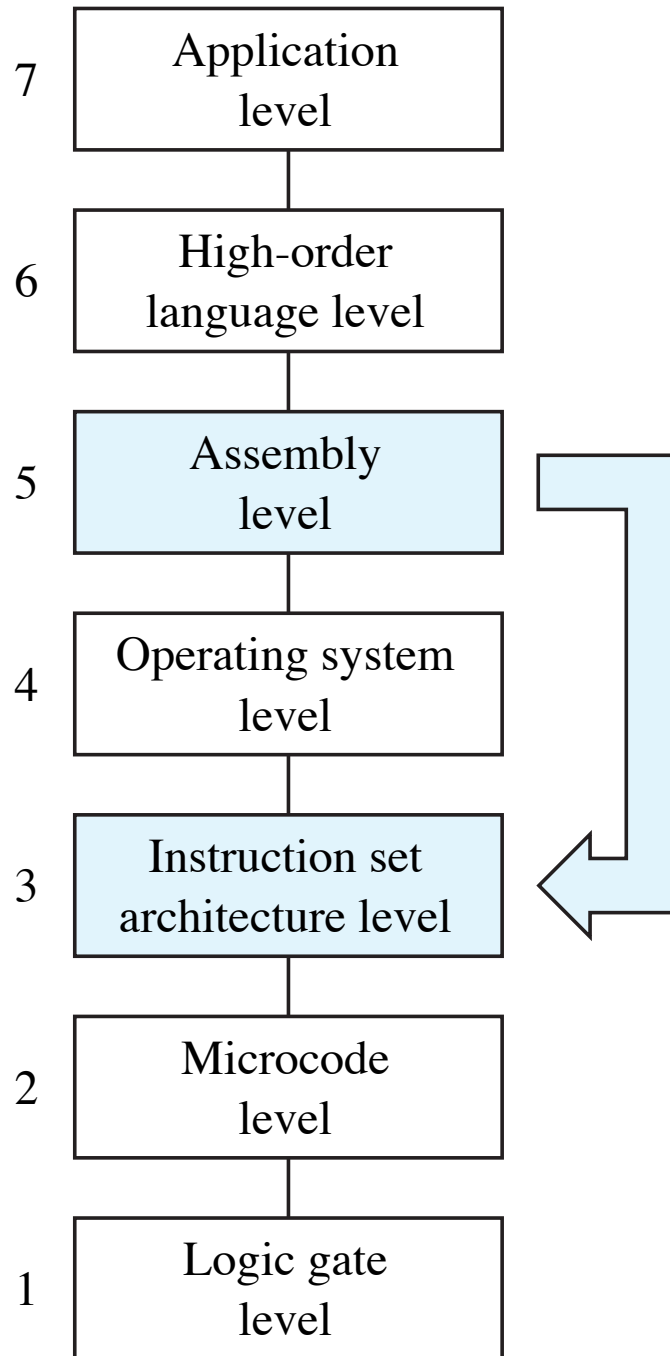
```
;Stan Warford
;January 13, 2005
;A program to output "Hi"
;
CHARO    0x0007,d    ;Output 'H'
CHARO    0x0008,d    ;Output 'i'
STOP
.ASCII   "Hi"
.END
```

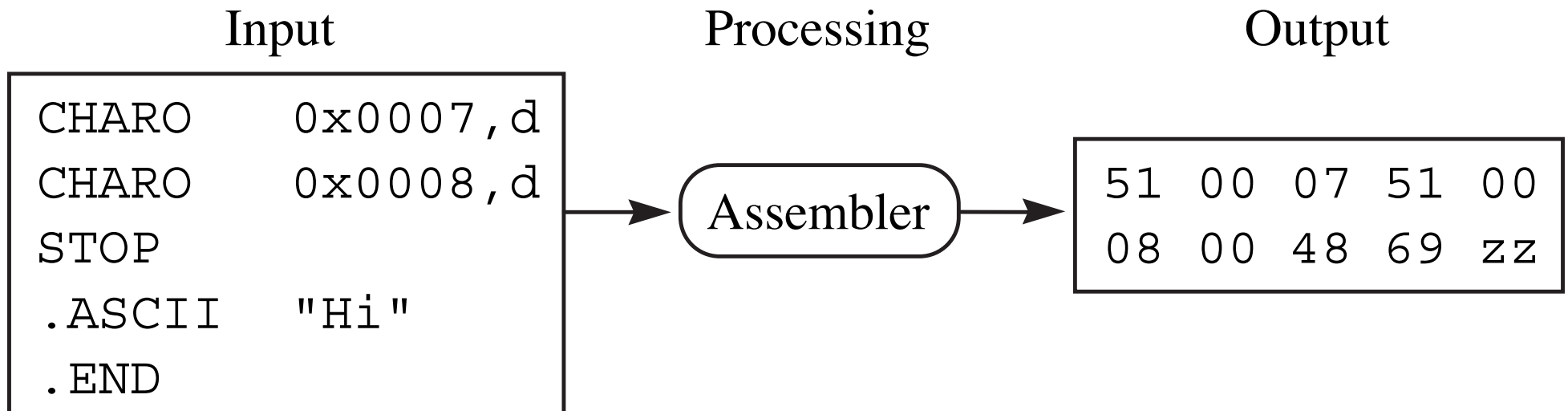
Assembler Output

```
51 00 07 51 00 08 00 48 69 zz
```

Program Output

```
Hi
```





Assembler Input

```
CHARI    0x000D,d      ;Input first character
CHARI    0x000E,d      ;Input second character
CHARO    0x000E,d      ;Output second character
CHARO    0x000D,d      ;Output first character
STOP
.BLOCK   1              ;Storage for first char
.BLOCK   1              ;Storage for second char
.END
```

Assembler Output

```
49 00 0D 49 00 0E 51 00 0E 51 00 0D 00 00 00 zz
```

Program Input

up

Program Output

pu

Assembler Input

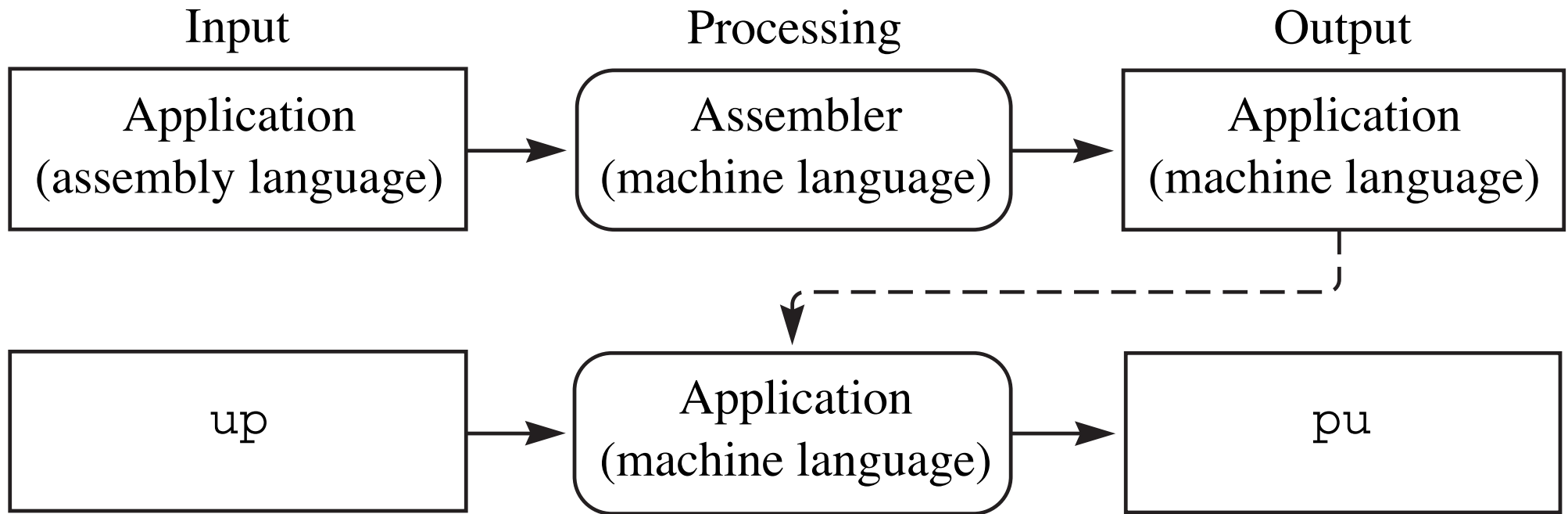
```
LDA      0x0011,d      ;A := first number
ADDA     0x0013,d      ;Add the two numbers
ORA      0x0015,d      ;Convert sum to character
STBYTEA 0x0010,d      ;Store the character
CHARO    0x0010,d      ;Output the character
STOP
.BLOCK   1             ;Character to output
.WORD    5             ;Decimal 5
.WORD    3             ;Decimal 3
.WORD    0x0030        ;Mask for ASCII char
.END
```

Assembler Output

```
C1 00 11 71 00 13 A1 00 15 F1 00 10 51 00 10 00
00 00 05 00 03 00 30 zz
```

Program Output

8



Assembler Input

```
chari 0x000D,d ;Input first character
CHARI 0x000E,d ;Input second character
charo 0x000E, d ;Output second character
CHARO 0x000D,D ;Output first character
stop
.block 1 ;Storage for first char
.BLOCK 1 ;Storage for second char
.END
```

Assembler Listing

Addr	code	Mnemon	Operand	Comment
0000	49000D	CHARI	0x000D,d	;Input first character
0003	49000E	CHARI	0x000E,d	;Input second character
0006	51000E	CHARO	0x000E,d	;Output second character
0009	51000D	CHARO	0x000D,d	;Output first character
000C	00	STOP		
000D	00	.BLOCK	1	;Storage for first char
000E	00	.BLOCK	1	;Storage for second char
000F		.END		

Direct addressing

- $\text{Oprnd} = \text{Mem}[\text{OprndSpec}]$
- Asmb5 letter: d
- The operand specifier is the *address* in memory of the operand.

Immediate addressing

- $\text{Oprnd} = \text{OprndSpec}$
- Asmb5 letter: *i*
- The operand specifier *is* the operand.

```
0000 500048 CHARO 'H',i ;Output 'H'  
0003 500069 CHARO 'i',i ;Output 'i'  
0006 00 STOP  
0007 .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

Oprnd \leftarrow {*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

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

The unconditional branch instruction

- Instruction specifier: 0000 010a
- Mnemonic: BR
- Skips to a different memory location for the next instruction to be executed.

$$PC \leftarrow \{\text{Oprnd}\}$$

```
0000 040005 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 500020 CHARO  ' ',i       ;Output " + 1 = "
000E 50002B CHARO  '+',i
0011 500020 CHARO  ' ',i
0014 500031 CHARO  '1',i
0017 500020 CHARO  ' ',i
001A 50003D CHARO  '=',i
001D 500020 CHARO  ' ',i
0020 C10003 LDA    0x0003,d     ;A := the number
0023 700001 ADDA   1,i         ;Add one to it
0026 E10003 STA    0x0003,d     ;Store the sum
0029 390003 DECO   0x0003,d     ;Output the sum
002C 00      STOP
002D      .END
```

Input

-479

Output

-479 + 1 = -478

The string output instruction

- Instruction specifier: 0100 0aaa
- Mnemonic: STRO
- Send a string of null-terminated ASCII characters to the output device

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

```
0000 04000D BR      0x000D      ;Branch around data
0003 0000  .BLOCK  2          ;Storage for one integer
0005 202B20 .ASCII  " + 1 = \x00"
      31203D
      2000

      ;
000D 310003 DECI   0x0003,d     ;Get the number
0010 390003 DECO   0x0003,d     ;and output it
0013 410005 STRO   0x0005,d     ;Output " + 1 = "
0016 C10003 LDA    0x0003,d     ;A := the number
0019 700001 ADDA   1,i         ;Add one to it
001C E10003 STA    0x0003,d     ;Store the sum
001F 390003 DECO   0x0003,d     ;Output the sum
0022 00      STOP
0023      .END
```

Input

-479

Output

-479 + 1 = -478

Interpreting bit patterns

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

```
0000 040009 BR      0x0009      ;Branch around data
0003 FFFE  .WORD   0xFFFFE     ;First
0005 00    .BYTE   0x00             ;Second
0006 55    .BYTE   'U'         ;Third
0007 0470  .WORD   1136        ;Fourth
;
0009 390003 DECO    0x0003,d     ;Interpret First as decimal
000C 50000A CHARO   '\n',i
000F 390005 DECO    0x0005,d     ;Interpret Second and Third as decimal
0012 50000A CHARO   '\n',i
0015 510006 CHARO   0x0006,d     ;Interpret Third as character
0018 510008 CHARO   0x0008,d     ;Interpret Fourth as character
001B 00    STOP
001C      .END
```

Output

-2

85

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 51000A CHARO 0x000A,d
0003 51000B CHARO 0x000B,d
0006 51000C CHARO 0x000C,d
0009 00        STOP
000A 50756E .ASCII "Pun"
000D          .END
```

Assembly Language Program

```
0000 51000A CHARO 0x000A,d
0003 51000B CHARO 0x000B,d
0006 51000C CHARO 0x000C,d
0009 00        STOP
000A 50756E CHARO 0x756E,i
000D          .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

Assembler Listing

Addr	Object code	Symbol	Mnemon	Operand	Comment
0000	04000D		BR	main	;Branch around data
0003	0000	num:	.BLOCK	2	;Storage for one integer
0005	202B20	msg:	.ASCII	" + 1 = \x00"	
	31203D				
	2000				
					;
000D	310003	main:	DECI	num,d	;Get the number
0010	390003		DECO	num,d	;and output it
0013	410005		STRO	msg,d	;Output ' + 1 = '
0016	C10003		LDA	num,d	;A := the number
0019	700001		ADDA	1,i	;Add one to it
001C	E10003		STA	num,d	;Store the sum
001F	390003		DECO	num,d	;Output the sum
0022	00		STOP		
0023			.END		

Symbol table:

Symbol	Value	Symbol	Value
main	000D	msg	0005
num	0003		

Input

-479

Output

-479 + 1 = -478

Assembler Input

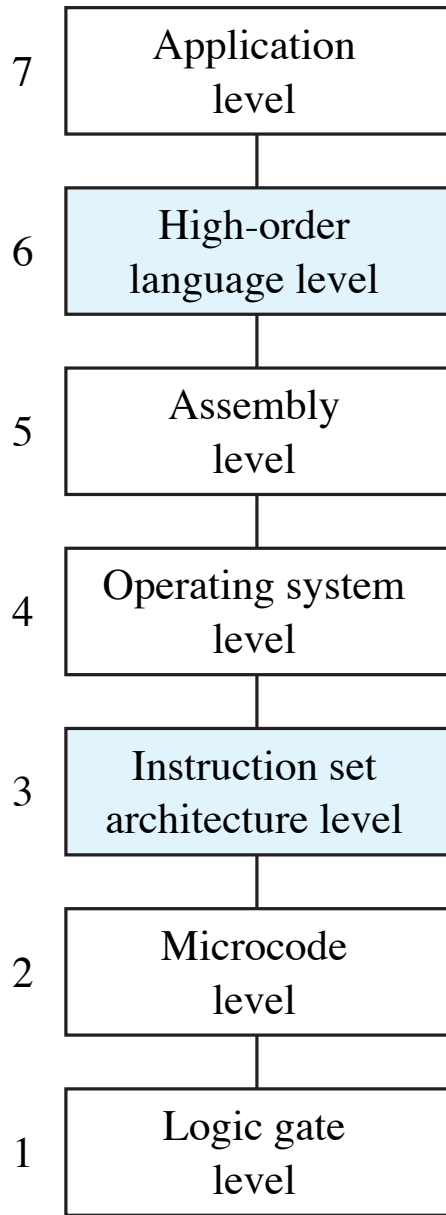
```
this: DECO this,d  
      STOP  
      .END
```

Assembler Listing

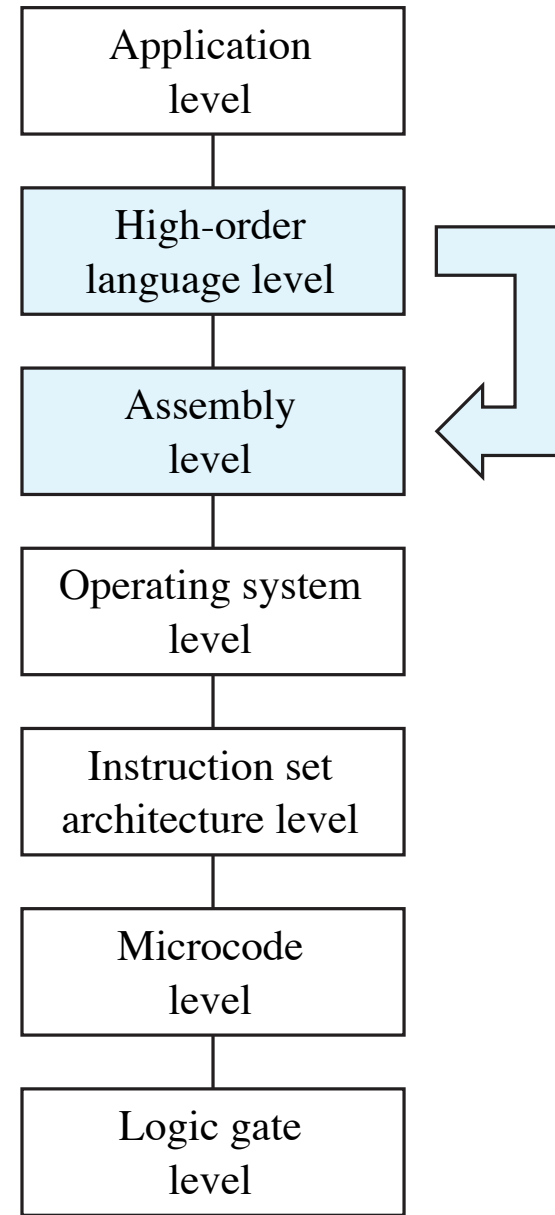
```
0000 390000 this:    DECO    this,d  
0003 00           STOP  
0004           .END
```

Output

```
14592
```



(a) Translation directly to machine language.



(b) Translation to assembly language.

Translating cout

- Translate string output with STRO
- Translate character output with CHARO
- Translate integer output with DECO

High-Order Language

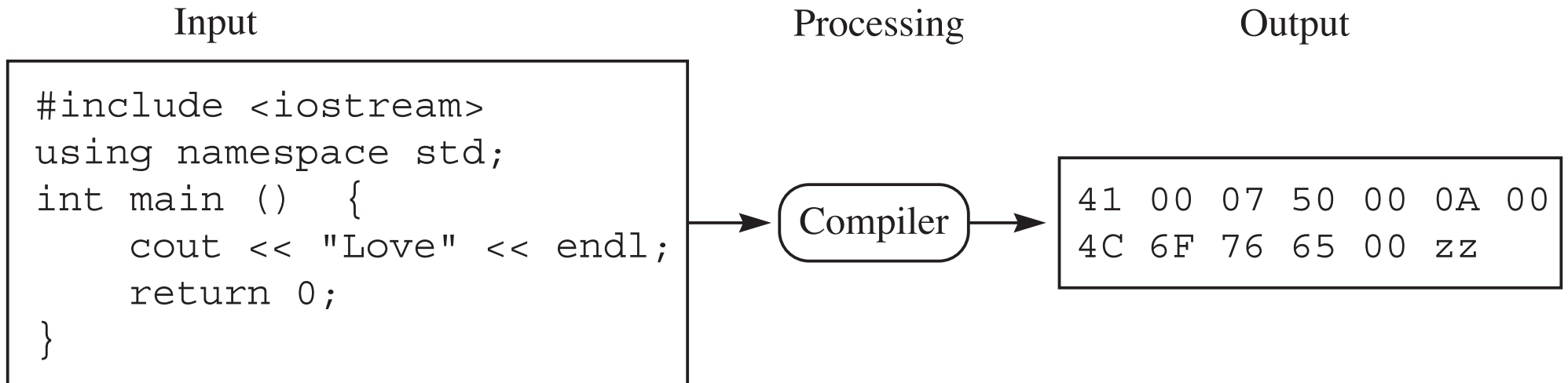
```
#include <iostream>
using namespace std;
int main () {
    cout << "Love" << endl;
    return 0;
}
```

Assembly Language

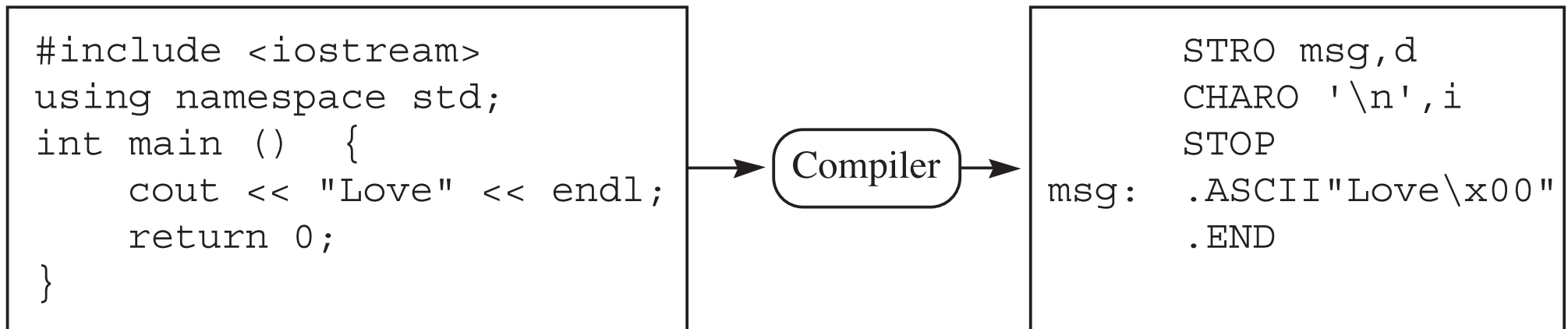
```
0000 410007          STRO    msg,d
0003 50000A          CHARO   '\n',i
0006 00              STOP
0007 4C6F76 msg:     .ASCII  "Love\x00"
        6500
000C              .END
```

Output

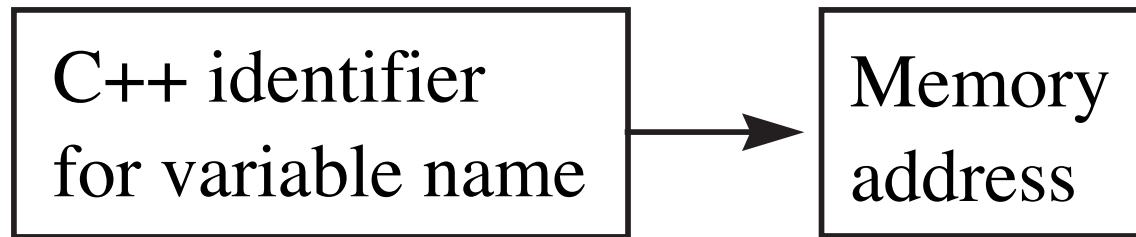
Love



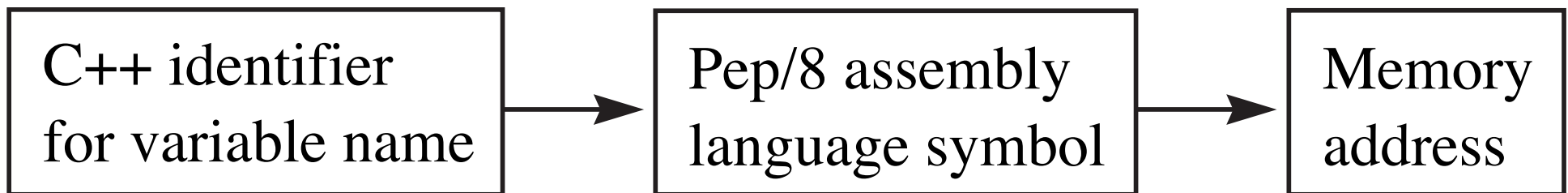
(a) A compiler that translates directly into machine language.



(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 LDA
- 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 STA

High-Order Language

```
#include <iostream>
using namespace std;

char ch;
int i;

int main () {
    cin >> ch >> i;
    i += 5;
    ch++;
    cout << ch << endl << i << endl;
    return 0;
}
```

Assembly Language

```
0000 040006          BR      main
0003 00      ch:    .BLOCK 1          ;global variable
0004 0000      i:    .BLOCK 2          ;global variable
        ;
0006 490003 main:    CHARI   ch,d          ;cin >> ch
0009 310004          DECI   i,d           ; >> i
000C C10004          LDA    i,d           ;i += 5
000F 700005          ADDA   5,i
0012 E10004          STA    i,d
0015 D10003          LDBYTEA ch,d         ;ch++
0018 700001          ADDA   1,i
001B F10003          STBYTEA ch,d
001E 510003          CHARO  ch,d         ;cout << ch
0021 50000A          CHARO  '\n',i        ; << endl
0024 390004          DECO   i,d           ; << i
0027 50000A          CHARO  '\n',i        ; << endl
002A 00          STOP
002B          .END
```

Input

M 419

Output

N

424

```
#include <iostream>
using namespace std;

char ch;
int i;

int main () {
    cin >> ch >> i;
    i += 5;
    ch++;
    cout << ch << endl << i << endl;
    return 0;
}
```

	symbol	value	kind
[0]	ch	0003	sChar
[1]	i	0004	sInt
[2]	:	:	:

```
#include <iostream>
using namespace std;

int j;
float y;

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

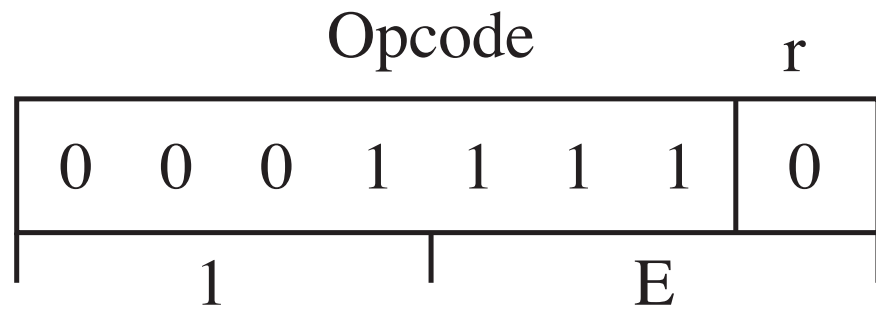
	symbol	value	kind
[0]	j	0003	sInt
[1]	y	0005	sFloat
[2]	:	:	:

The arithmetic shift right instruction

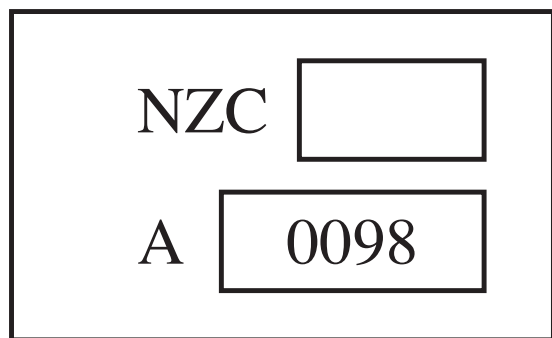
- Instruction specifier: 0001 111r
- Mnemonic: ASRr (ASRA, ASRX)
- Performs a one-bit arithmetic shift right on a 16-bit register

$$C \leftarrow r\langle 15 \rangle, r\langle 1..15 \rangle \leftarrow r\langle 0..14 \rangle;$$
$$N \leftarrow r < 0, Z \leftarrow r = 0$$

Instruction specifier



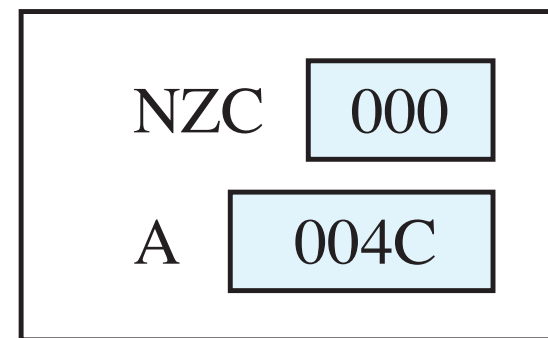
CPU



(a) Before

1E
Arithmetic shift right
accumulator

CPU



(b) After

The arithmetic shift left instruction

- Instruction specifier: 0001 110r
- Mnemonic: ASLr (ASLA, ASLX)
- Performs a one-bit arithmetic shift left on a 16-bit register

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

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

The rotate left instruction

- Instruction specifier: 0010 000r
- 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: 0010 001r
- Mnemonic: RORr (RORA, RORX)
- Performs a one-bit rotate right on a 16-bit register

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

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 <iostream>
using namespace std;

const int bonus = 5;
int exam1;
int exam2;
int score;

int main () {
    cin >> exam1 >> exam2;
    score = (exam1 + exam2) / 2 + bonus;
    cout << "score = " << score << endl;
    return 0;
}
```

Assembly Language

```
0000 040009          BR      main
                   bonus:  .EQUATE 5          ;constant
0003 0000  exam1:    .BLOCK 2          ;global variable
0005 0000  exam2:    .BLOCK 2          ;global variable
0007 0000  score:    .BLOCK 2          ;global variable
                   ;
0009 310003 main:    DECI    exam1,d      ;cin >> exam1
000C 310005          DECI    exam2,d      ;  >> exam2
000F C10003          LDA     exam1,d      ;score = (exam1
0012 710005          ADDA   exam2,d      ;  + exam2)
0015 1E             ASRA           ;  / 2
0016 700005          ADDA   bonus,i      ;  + bonus
0019 E10007          STA     score,d
001C 410026          STRO   msg,d      ;cout << "score = "
001F 390007          DECO   score,d      ;  << score
0022 50000A          CHARO  '\n',i      ;  << endl
0025 00             STOP
0026 73636F msg:    .ASCII  "score = \x00"
                   726520
                   3D2000
002F                .END
```

Symbol table:

Symbol	Value	Symbol	Value
bonus	0005	exam1	0003
exam2	0005	main	0009
msg	0026	score	0007

Input

68 84

Output

score = 81

Assembly Language

```

0000 31001D main:    DECI    exam1,d        ;cin >> exam1
0003 31001F          DECI    exam2,d        ;  >> exam2
0006 C1001D          LDA     exam1,d        ;score = (exam1
0009 71001F          ADDA   exam2,d        ;  + exam2)
000C 1E              ASRA                   ;  / 2
000D 700005          ADDA   bonus,i        ;  + bonus
0010 E10021          STA     score,d
0013 410023          STRO   msg,d          ;cout << "score = "
0016 390021          DECO   score,d        ;  << score
0019 50000A          CHARO  '\n',i        ;  << endl
001C 00              STOP

;
bonus: .EQUATE 5        ;constant
001D 0000 exam1: .BLOCK 2        ;global variable
001F 0000 exam2: .BLOCK 2        ;global variable
0021 0000 score: .BLOCK 2        ;global variable
0023 73636F msg: .ASCII "score = \x00"
      726520
      3D2000

002C .END

```

