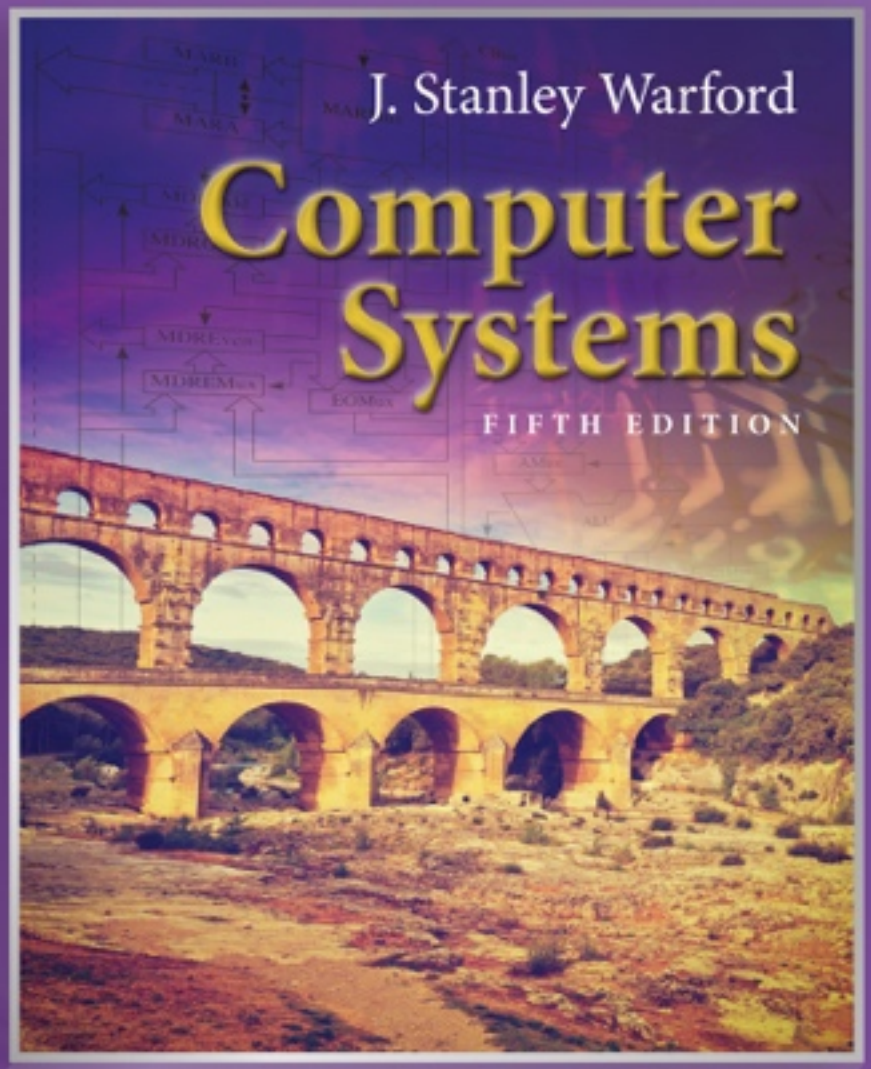
Chapter 6

Compiling to the Assembly Level



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.

Stack-relative addressing

- Oprnd = Mem[SP + OprndSpec]
- Asmb5 letter: `s`
- The stack pointer *plus* the operand specifier is the *address* in memory of the operand.

The add SP instruction

- Instruction specifier: 0101 0aaa
- Mnemonic: `ADDSP`
- Adds the operand to the stack pointer, SP

$$SP \leftarrow SP + Oprnd$$

The subtract SP instruction

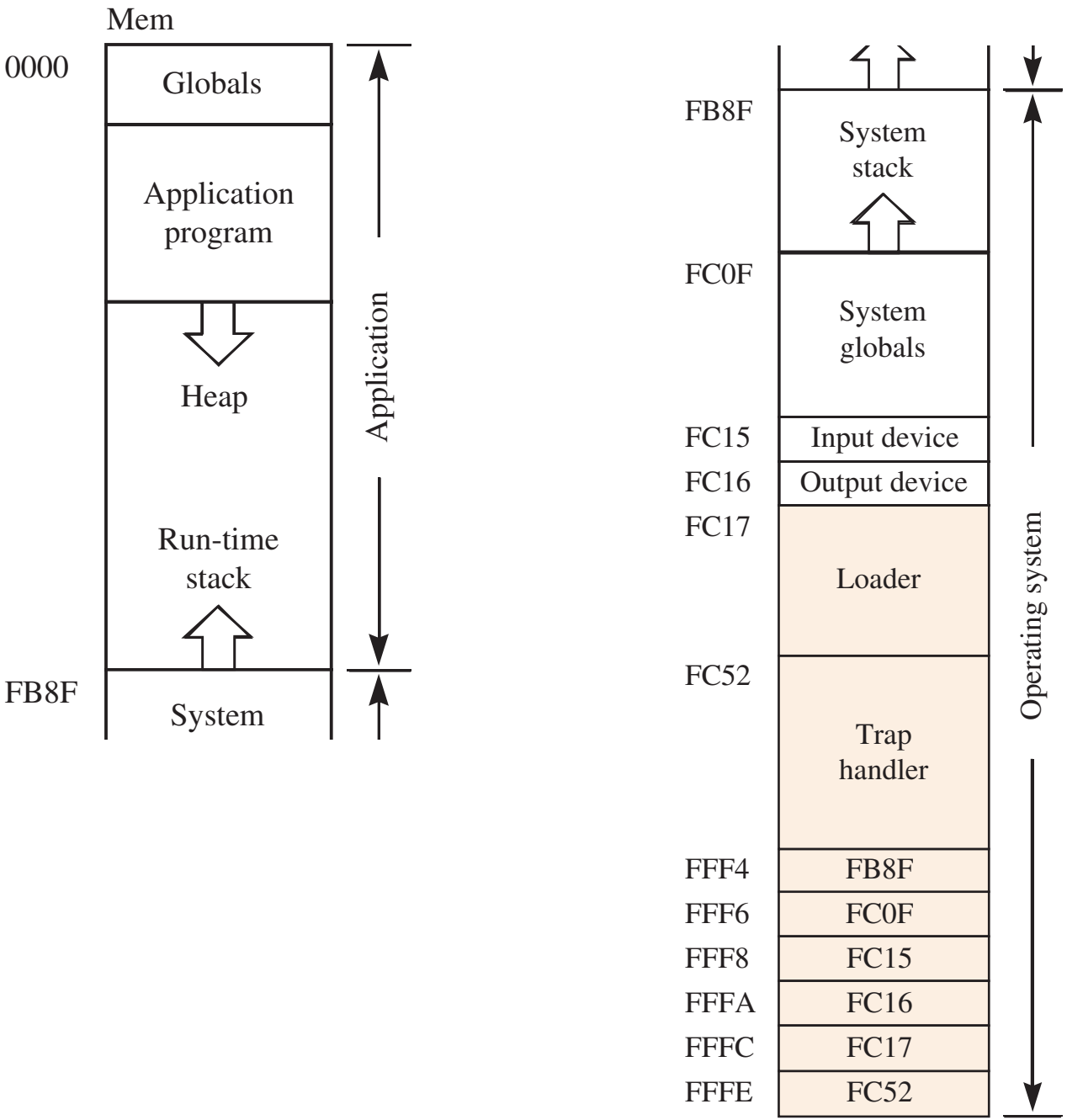
- Instruction specifier: 0101 1aaa
- Mnemonic: `SUBSP`
- Subtracts the operand from the stack pointer, SP

$$SP \leftarrow SP - Oprnd$$

Pep/9 execute option

- SP←Mem[FFF4]
- PC←0000

Figure 4.41

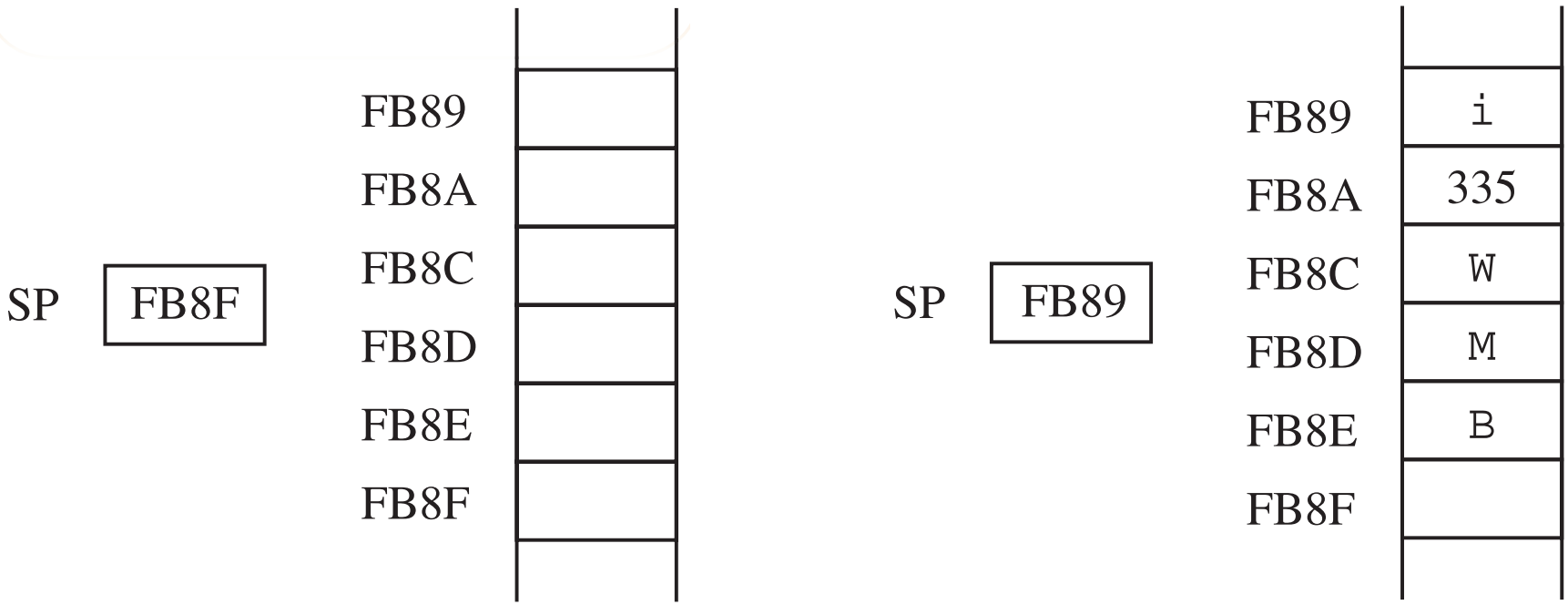


```
0000  D00042    LDBA    'B',i        ;move B to stack
0003  F3FFFF    STBA    -1,s
0006  D0004D    LDBA    'M',i        ;move M to stack
0009  F3FFFE    STBA    -2,s
000C  D00057    LDBA    'W',i        ;move W to stack
000F  F3FFFD    STBA    -3,s
0012  C0014F    LDWA    335,i        ;move 335 to stack
0015  E3FFFB    STWA    -5,s
0018  D00069    LDBA    'i',i        ;move i to stack
001B  F3FFFA    STBA    -6,s
001E  580006    SUBSP   6,i          ;push 6 bytes onto stack
0021  D30005    LDBA    5,s          ;output B
0024  F1FC16    STBA    charOut,d
0027  D30004    LDBA    4,s          ;output M
002A  F1FC16    STBA    charOut,d
002D  D30003    LDBA    3,s          ;output W
0030  F1FC16    STBA    charOut,d
0033  3B0001    DECO    1,s          ;output 335
0036  D30000    LDBA    0,s          ;output i
0039  F1FC16    STBA    charOut,d
003C  500006    ADDSP   6,i          ;pop 6 bytes off stack
003F  00        STOP
0040            .END
```

Output

```
BMW335i
```

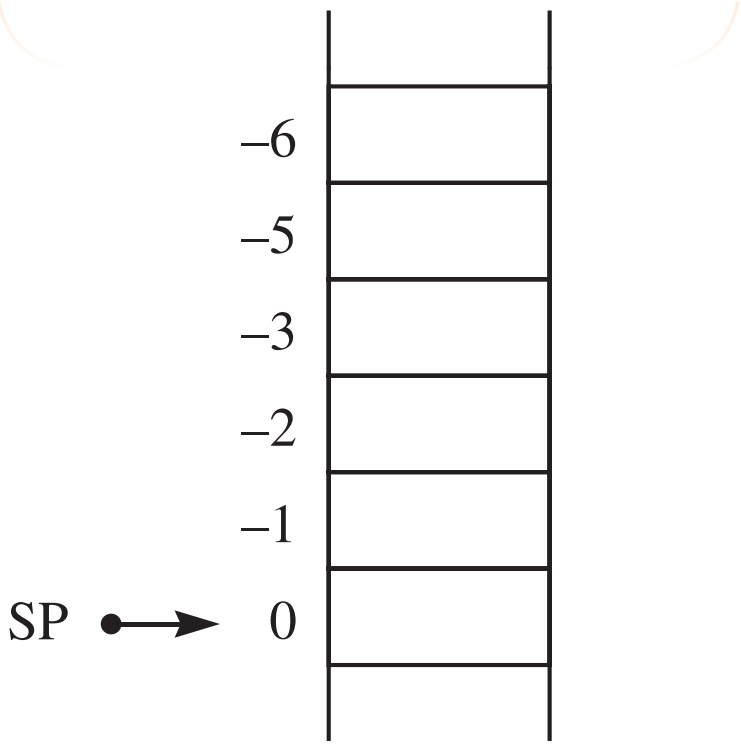
Absolute addresses



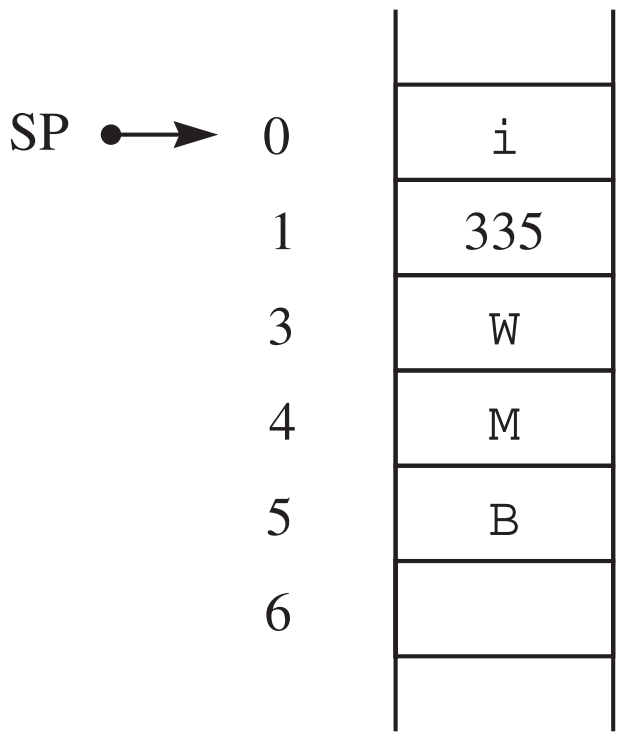
(a) Before the program executes.

(b) After `SUBSP` executes.

Stack-relative addresses



(a) Before the program executes.



(b) After `SUBSP` executes.

Local variables

- Push locals with `SUBSP`
- Access locals with stack-relative addressing (`s`)
- Pop locals with `ADDSP`

High-Order Language

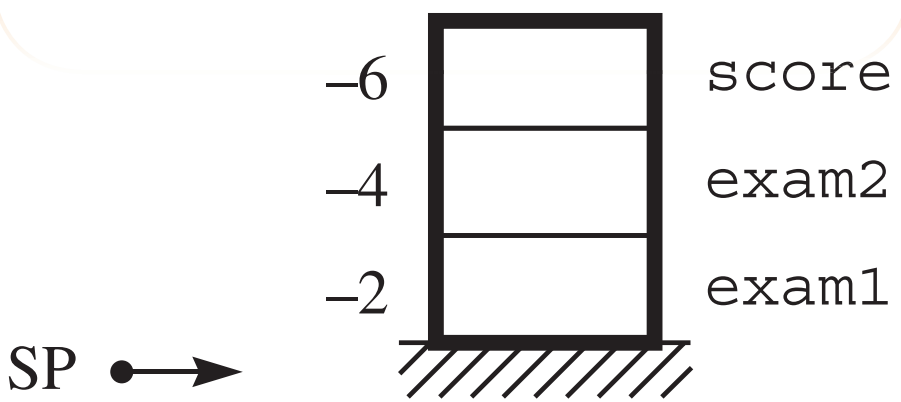
```
#include <stdio.h>

int main() {
   const int bonus = 10;
   int exam1;
   int exam2;
   int score;
   scanf("%d %d", &exam1, &exam2);
   score = (exam1 + exam2) / 2 + bonus;
   printf("score = %d\n", score);
   return 0;
}
```

Figure 6.4 (continued)

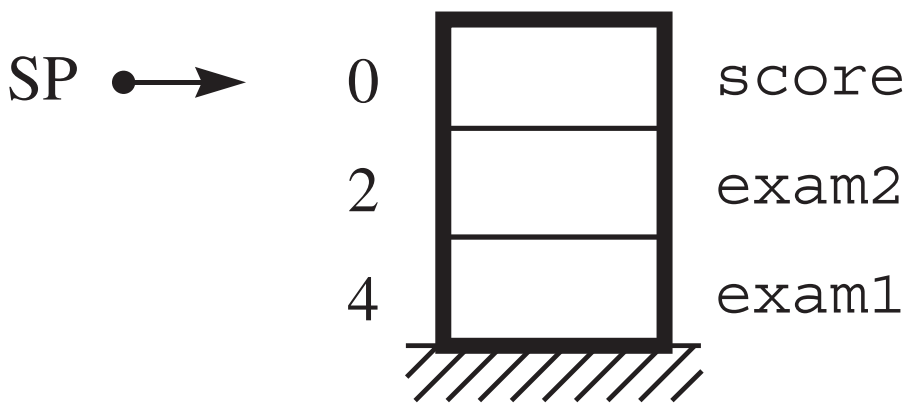
Assembly Language

```
0000   120003           BR      main
               bonus:   .EQUATE 10          ;constant
               exam1:   .EQUATE 4           ;local variable #2d
               exam2:   .EQUATE 2           ;local variable #2d
               score:   .EQUATE 0           ;local variable #2d
               ;
0003   580006 main:     SUBSP   6,i          ;push #exam1 #exam2 #score
0006   330004           DECI    exam1,s      ;scanf("%d %d", &exam1, &exam2)
0009   330002           DECI    exam2,s
000C   C30004           LDWA    exam1,s      ;score = (exam1 + exam2) / 2 + bonus
000F   630002           ADDA    exam2,s
0012   0C               ASRA
0013   60000A           ADDA    bonus,i
0016   E30000           STWA    score,s
0019   490029           STRO    msg,d        ;printf("score = %d\n", score)
001C   3B0000           DECO    score,s
001F   D0000A           LDBA    '\n',i
0022   F1FC16           STBA    charOut,d
0025   500006           ADDSP   6,i          ;pop #score #exam2 #exam1
0028   00               STOP
0029   73636F msg:      .ASCII  "score = \x00"
       726520
       3D2000
0032                    .END
```

Figure 6.5



(a) Before `SUBSP` executes.



(b) After `SUBSP` executes.

Branching instructions

- `BRLE` Branch on less than or equal to
- `BRLT` Branch on less than
- `BREQ` Branch on equal to
- `BRNE` Branch on not equal to
- `BRGE` Branch on greater than or equal to
- `BRGT` Branch on greater than
- `BRV` Branch on V
- `BRC` Branch on C

Branching instructions

- `BRLE` $N = 1 \vee Z = 1 \Rightarrow PC \leftarrow Oprnd$
- `BRLT` $N = 1 \Rightarrow PC \leftarrow Oprnd$
- `BREQ` $Z = 1 \Rightarrow PC \leftarrow Oprnd$
- `BRNE` $Z = 0 \Rightarrow PC \leftarrow Oprnd$
- `BRGE` $N = 0 \Rightarrow PC \leftarrow Oprnd$
- `BRGT` $N = 0 \wedge Z = 0 \Rightarrow PC \leftarrow Oprnd$
- `BRV` $V = 1 \Rightarrow PC \leftarrow Oprnd$
- `BRC` $C = 1 \Rightarrow PC \leftarrow Oprnd$

High-Order Language

```
#include <stdio.h>

int main() {
    int number;
    scanf("%d", &number);
    if (number < 0) {
        number = -number;
    }
    printf("%d", number);
    return 0;
}
```

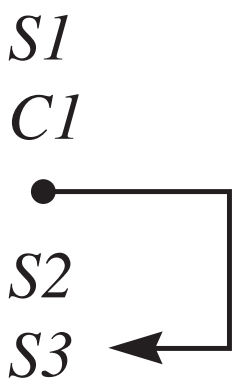
Figure 6.6 (continued)

Assembly Language

```
0000   120003           BR      main
                number:  .EQUATE 0           ;local variable #2d
                ;
0003   580002 main:      SUBSP   2,i          ;push #number
0006   330000           DECI    number,s     ;scanf("%d", &number)
0009   C30000 if:        LDWA    number,s     ;if (number < 0)
000C   1C0016           BRGE    endIf
000F   C30000           LDWA    number,s     ;number = -number
0012   08               NEGA
0013   E30000           STWA    number,s
0016   3B0000 endIf:    DECO    number,s     ;printf("%d", number)
0019   500002           ADDSP   2,i          ;pop #number
001C   00               STOP
001D                    .END
```

```
S1
if (C1) {
   S2
}
S3
```

(a) The structure at Level HOL6.



(b) The same structure at Level Asmb5.

Optimizing compiler

- Eliminates unnecessary instructions in the object code
- Advantage: Object code runs faster
- Disadvantage: Takes longer to compile

Compare word instruction

- Instruction specifier: 1010 raaa
- Mnemonic: `CPWr` (`CPWA`, `CPWX`)
- Compare r with operand

$$T \leftarrow r - Oprnd\ ;\ N \leftarrow T < 0\ ,\ Z \leftarrow T = 0\ ,$$
$$V \leftarrow \{overflow\}\ ,\ C \leftarrow \{carry\}\ ;\ N \leftarrow N \oplus V$$

High-Order Language

```
#include <stdio.h>

int main() {
   const int limit = 100;
   int num;
   scanf("%d", &num);
   if (num >= limit) {
      printf("high\n");
   }
   else {
      printf("low\n");
   }
   return 0;
}
```

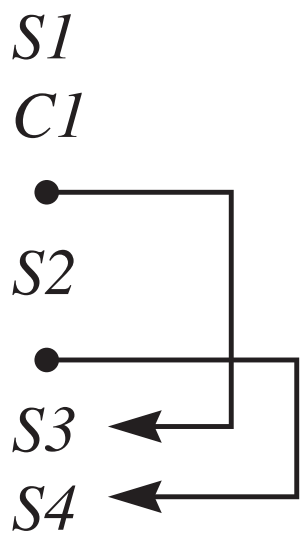
Figure 6.8
(continued)

Assembly Language

```
0000  120003          BR      main
                limit:  .EQUATE 100         ;constant
                num:    .EQUATE 0           ;local variable #2d
                ;
0003  580002  main:     SUBSP   2,i         ;push #num
0006  330000          DECI    num,s         ;scanf("%d", &num)
0009  C30000  if:       LDWA    num,s         ;if (num >= limit)
000C  A00064          CPWA    limit,i
000F  160018          BRLT    else
0012  49001F          STRO    msg1,d        ;printf("high\n")
0015  12001B          BR      endIf
0018  490025  else:     STRO    msg2,d        ;printf("low\n")
001B  500002  endIf:    ADDSP   2,i           ;pop #num
001E  00              STOP
001F  686967  msg1:     .ASCII  "high\n\x00"
      680A00
0025  6C6F77  msg2:     .ASCII  "low\n\x00"
      0A00
002A                  .END
```

```
S1
if (C1) {
   S2
}
else {
   S3
}
S4
```

(a) The structure at Level HOL6.



(b) The same structure at Level Asmb5.

High-Order Language

```
#include <stdio.h>

char letter;

int main() {
   scanf("%c", &letter);
   while (letter != '*') {
      if (letter == ' ') {
         printf("\n");
      }
      else {
         printf("%c", letter);
      }
      scanf("%c", &letter);
   }
   return 0;
}
```

Figure 6.10 (continued)

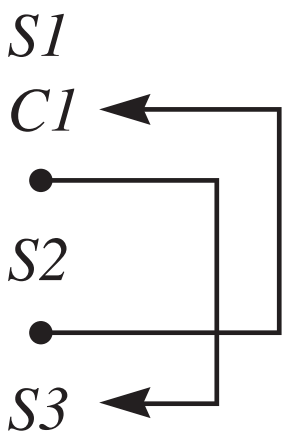
Assembly Language

```
0000  120004           BR      main
0003  00      letter:  .BLOCK  1           ;global variable #1c
              ;
0004  D1FC15  main:    LDBA    charIn,d    ;scanf("%c", &letter)
0007  F10003           STBA    letter,d
000A  D10003  while:   LDBA    letter,d    ;while (letter != '*')
000D  B0002A           CPBA    '*',i
0010  18002E           BREQ    endWh
0013  B00020  if:      CPBA    ' ',i       ;if (letter == ' ')
0016  1A0022           BRNE    else
0019  D0000A           LDBA    '\n',i      ;printf("\n")
001C  F1FC16           STBA    charOut,d
001F  120025           BR      endIf
0022  F1FC16  else:    STBA    charOut,d   ;printf("%c", letter)
0025  D1FC15  endIf:   LDBA    charIn,d    ;scanf("%c", &letter)
0028  F10003           STBA    letter,d
002B  12000A           BR      while
002E  00      endWh:   STOP
002F                   .END
```

Figure 6.11

```
S1
while (C1) {
   S2
}
S3
```

(a) The structure at Level HOL6.



(b) The same structure at Level Asmb5.

High-Order Language

```
#include <stdio.h>

int cop;
int driver;

int main() {
   cop = 0;
   driver = 40;
   do {
      cop += 25;
      driver += 20;
   }
   while (cop < driver);
   printf("%d", cop);
   return 0;
}
```

Figure 6.12 (continued)

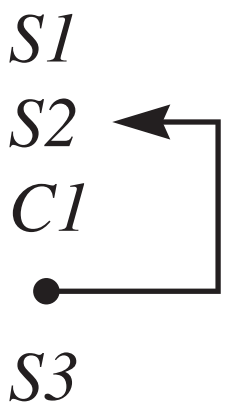
Assembly Language

```
0000  120007           BR      main
0003  0000    cop:     .BLOCK  2           ;global variable #2d
0005  0000    driver:  .BLOCK  2           ;global variable #2d
              ;
0007  C00000  main:    LDWA    0,i         ;cop = 0
000A  E10003           STWA    cop,d
000D  C00028           LDWA    40,i        ;driver = 40
0010  E10005           STWA    driver,d
0013  C10003  do:      LDWA    cop,d       ;cop += 25
0016  600019           ADDA    25,i
0019  E10003           STWA    cop,d
001C  C10005           LDWA    driver,d    ;driver += 20
001F  600014           ADDA    20,i
0022  E10005           STWA    driver,d
0025  C10003  while:   LDWA    cop,d       ;while (cop < driver)
0028  A10005           CPWA    driver,d
002B  160013           BRLT    do
002E  390003           DECO    cop,d       ;printf("%d", cop)
0031  00               STOP
0032                   .END
```

Figure 6.13

```
S1
do {
    S2
}
while (C1)
S3
```

(a) The structure at Level HOL6.



(b) The same structure at Level Asmb5.

The `for` loop

- Initialize the control variable
- Test the control variable
- Execute the loop body
- Increment the control variable
- Branch to the test

High-Order Language

```
#include <stdio.h>

int main() {
   int j;
   for (j = 0; j < 3; j++) {
      printf("j = %d\n", j);
   }
   return 0;
}
```

Figure 6.14 (continued)

Assembly Language

```
0000  120003          BR      main
              j:      .EQUATE 0           ;local variable #2d
              ;
0003  580002  main:   SUBSP   2,i         ;push #j
0006  C00000          LDWA    0,i         ;for (j = 0
0009  E30000          STWA    j,s
000C  A00003  for:    CPWA    3,i         ;j < 3
000F  1C002A          BRGE    endFor
0012  49002E          STRO    msg,d       ;printf("j = %d\n", j)
0015  3B0000          DECO    j,s
0018  D0000A          LDBA    '\n',i
001B  F1FC16          STBA    charOut,d
001E  C30000          LDWA    j,s         ;j++)
0021  600001          ADDA    1,i
0024  E30000          STWA    j,s
0027  12000C          BR      for
002A  500002  endFor: ADDSP   2,i         ;pop #j
002D  00              STOP
002E  6A203D  msg:    .ASCII  "j = \x00"
      2000
0033                  .END
```

Not possible in many HOL6 languages

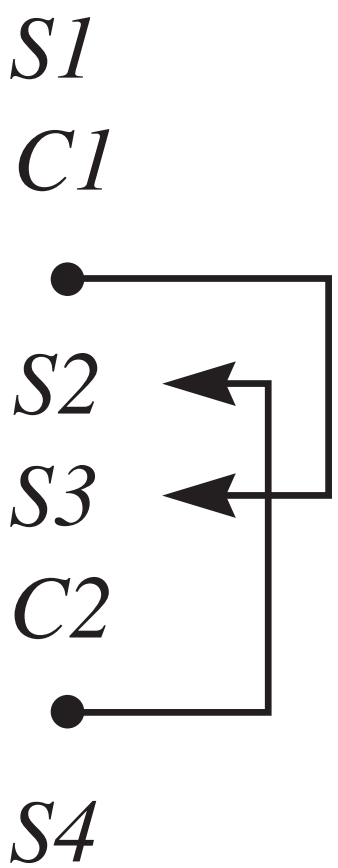
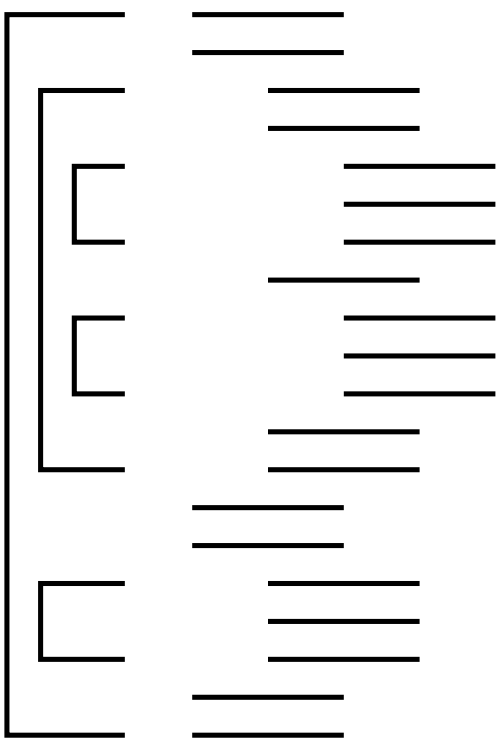


Figure 6.16

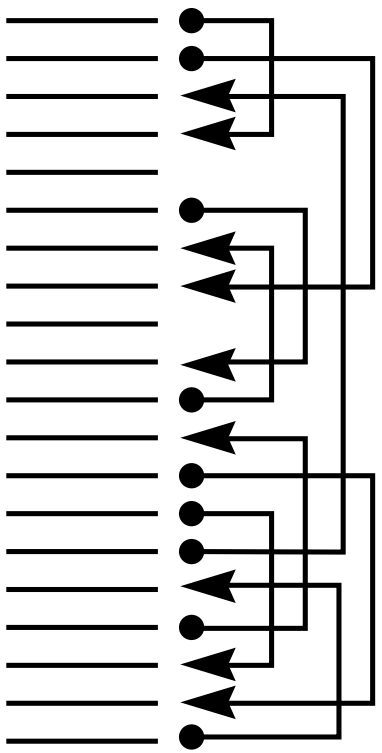
```
0000  120009          BR      main
0003  0000    n1:     .BLOCK  2           ;#2d
0005  0000    n2:     .BLOCK  2           ;#2d
0007  0000    n3:     .BLOCK  2           ;#2d
              ;
0009  310005  main:   DECI    n2,d
000C  310007          DECI    n3,d
000F  C10005          LDWA    n2,d
0012  A10007          CPWA    n3,d
0015  16002A          BRLT    L1
0018  310003          DECI    n1,d
001B  C10003          LDWA    n1,d
001E  A10007          CPWA    n3,d
0021  160074          BRLT    L7
0024  120065          BR      L6
0027  E10007          STWA    n3,d
002A  310003  L1:     DECI    n1,d
002D  C10005          LDWA    n2,d
0030  A10003          CPWA    n1,d
0033  160053          BRLT    L5
0036  390003          DECO    n1,d
0039  390005          DECO    n2,d
003C  390007  L2:     DECO    n3,d
003F  00              STOP
```

Figure 6.16
(continued)

```
0040  390005 L3:     DECO    n2,d
0043  390007         DECO    n3,d
0046  120081         BR      L9
0049  390003 L4:     DECO    n1,d
004C  390005         DECO    n2,d
004F  00             STOP
0050  E10003         STWA    n1,d
0053  C10007 L5:     LDWA    n3,d
0056  A10003         CPWA    n1,d
0059  160040         BRLT    L3
005C  390005         DECO    n2,d
005F  390003         DECO    n1,d
0062  12003C         BR      L2
0065  390007 L6:     DECO    n3,d
0068  C10003         LDWA    n1,d
006B  A10005         CPWA    n2,d
006E  160049         BRLT    L4
0071  12007E         BR      L8
0074  390003 L7:     DECO    n1,d
0077  390007         DECO    n3,d
007A  390005         DECO    n2,d
007D  00             STOP
007E  390005 L8:     DECO    n2,d
0081  390003 L9:     DECO    n1,d
0084  00             STOP
0085                 .END
```



(a) Structured flow.



(b) Spaghetti code.

The Structured Programming Theorem

Any algorithm containing goto's, no matter how complicated or unstructured, can be written with only nested if statements and while loops.

Bohm and Jacopini, 1966

The goto controversy

More recently I discovered why the use of the goto statement has such disastrous effects, and I became convinced that the goto statement should be abolished from all "higher level" programming languages. ... The goto statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program.

Dijkstra, 1968

The call subroutine instruction

- Instruction specifier: 0010 010a
- Mnemonic: `CALL`
- Push return address onto run-time stack

$$SP \leftarrow SP - 2 \; ; \; Mem[SP] \leftarrow PC \; ; \; PC \leftarrow Oprnd$$

The return from subroutine instruction

- Instruction specifier: 0000 0001
- Mnemonic: `RET`
- Pop return address off of run-time stack

$$PC \leftarrow Mem[SP] \; ; \; SP \leftarrow SP + 2$$

High-Order Language

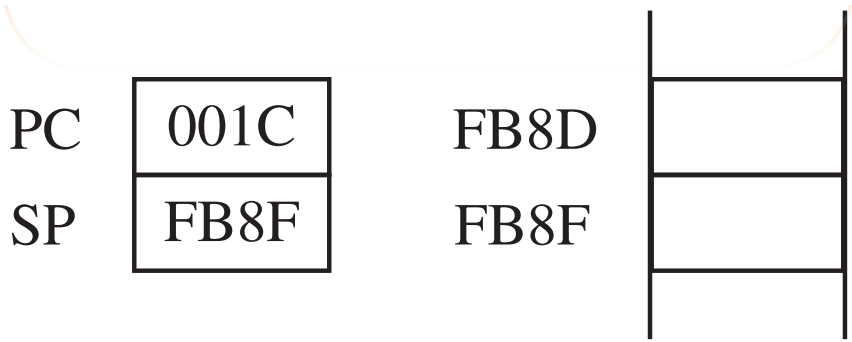
```
#include <stdio.h>

void printTri() {
   printf("*\n");
   printf("**\n");
   printf("***\n");

}

int main() {
   printTri();
   printTri();
   printTri();
   return 0;
}
```

Figure 6.18 (continued)

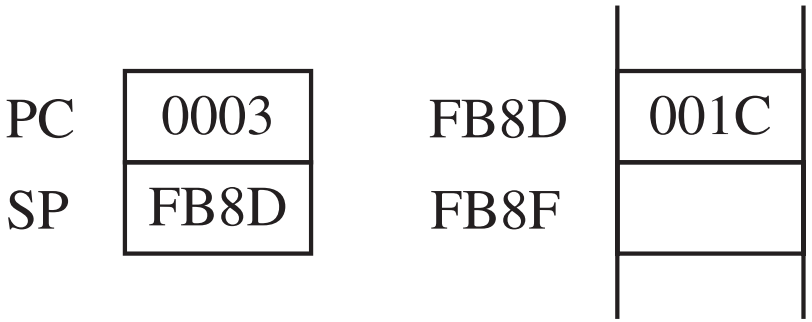
Assembly Language

```
000  120019          BR      main
                ;
                ;******* void printTri()
0003  49000D printTri:STRO    msg1,d      ;printf("*\n")
0006  490010          STRO    msg2,d      ;printf("**\n")
0009  490014          STRO    msg3,d      ;printf("***\n")
000C  01              RET
000D  2A0A00 msg1:    .ASCII  "*\n\x00"
0010  2A2A0A msg2:    .ASCII  "**\n\x00"
      00
0014  2A2A2A msg3:    .ASCII  "***\n\x00"
      0A00
                ;
                ;******* int main()
0019  240003 main:    CALL    printTri    ;printTri()
001C  240003          CALL    printTri    ;printTri()
001F  240003          CALL    printTri    ;printTri()
0022  00              STOP
0023                  .END
```

Figure 6.19

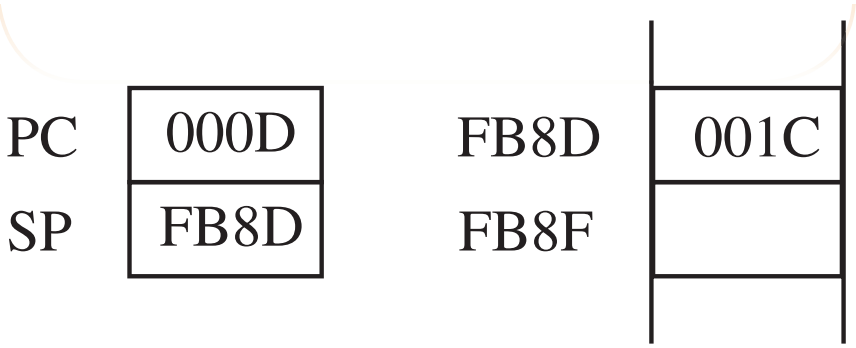


(a) Before execution of the first `CALL`.

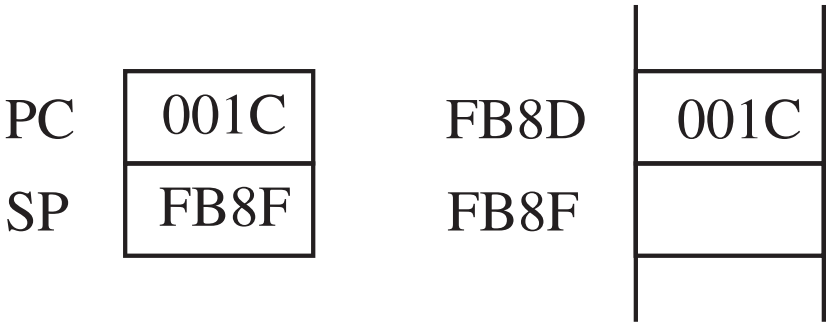


(b) After execution of the first `CALL`.

Figure 6.20



(a) Before the first execution of `RET`.



(b) After the first execution of `RET`.

To call a `void` function in C

- Push the actual parameters
- Push the return address
- Push storage for the local variables

In assembly language

- Caller pushes actual parameters (executes `SUBSP`)
- Caller pushes return address (executes `CALL`)
- Callee allocates local variables (executes `SUBSP`)
- Callee executes its body.
- Callee pops local variables (executes `ADDSP`)
- Callee pops return address (executes `RET`)
- Caller pops actual parameters (executes `ADDSP`)

Call-by-value with global variables

- To get the actual parameter in the caller, generate load with direct addressing (d)
- To get the formal parameter in the callee, generate load with stack-relative addressing (s)

High-Order Language

```
#include <stdio.h>

int numPts;
int value;
int j;

void printBar(int n) {
   int k;
   for (k = 1; k <= n; k++) {
      printf("*");
   }
   printf("\n");
}

int main() {
   scanf("%d", &numPts);
   for (j = 1; j <= numPts; j++) {
      scanf("%d", &value);
      printBar(value);
   }
   return 0;
}
```

Figure 6.21 (continued)

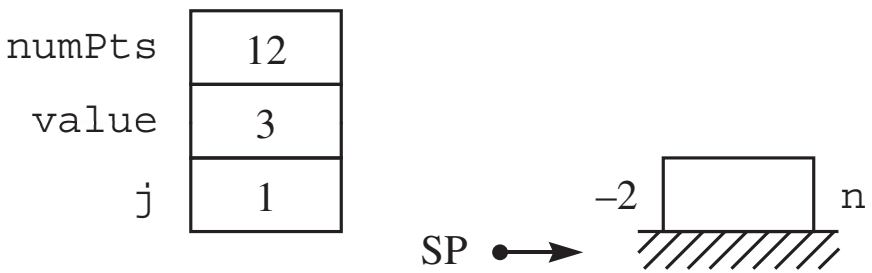
Assembly Language

```
0000  120034          BR      main
0003  0000    numPts: .BLOCK  2           ;global variable #2d
0005  0000    value:  .BLOCK  2           ;global variable #2d
0007  0000    j:      .BLOCK  2           ;global variable #2d
              ;
              ;******* void printBar(int n)
              n:      .EQUATE 4           ;formal parameter #2d
              k:      .EQUATE 0           ;local variable #2d
0009  580002  printBar:SUBSP  2,i         ;push #k
000C  C00001          LDWA    1,i         ;for (k = 1
000F  E30000          STWA    k,s
0012  A30004  for1:   CPWA    n,s         ;k <= n
0015  1E002A          BRGT    endFor1
0018  D0002A          LDBA    '*',i       ;printf("*")
001B  F1FC16          STBA    charOut,d
001E  C30000          LDWA    k,s         ;k++)
0021  600001          ADDA    1,i
0024  E30000          STWA    k,s
0027  120012          BR      for1
002A  D0000A  endFor1:LDBA    '\n',i      ;printf("\n")
002D  F1FC16          STBA    charOut,d
0030  500002          ADDSP   2,i         ;pop #k
0033  01              RET
```

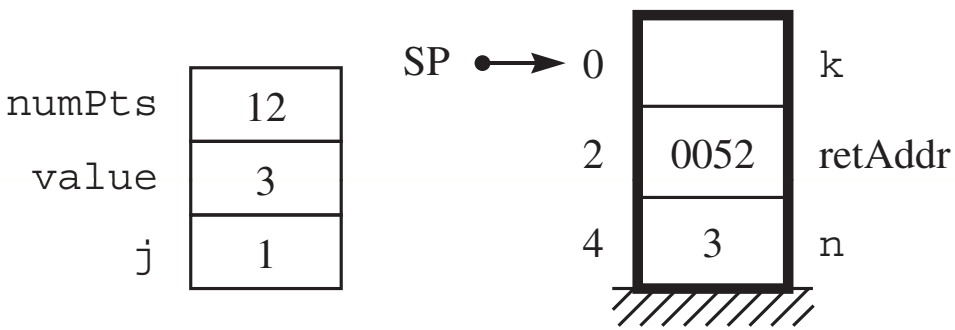
Figure 6.21 (continued)

Assembly Language

```
                ;******* main()
0034   310003 main:    DECI    numPts,d    ;scanf("%d", &numPts)
0037   C00001           LDWA    1,i         ;for (j = 1
003A   E10007           STWA    j,d
003D   A10003 for2:     CPWA    numPts,d    ;j <= numPts
0040   1E0061           BRGT    endFor2
0043   310005           DECI    value,d     ;scanf("%d", &value)
0046   C10005           LDWA    value,d     ;move value
0049   E3FFFE           STWA    -2,s
004C   580002           SUBSP   2,i         ;push #n
004F   240009           CALL    printBar    ;printBar(value)
0052   500002           ADDSP   2,i         ;pop #n
0055   C10007           LDWA    j,d         ;j++)
0058   600001           ADDA    1,i
005B   E10007           STWA    j,d
005E   12003D           BR      for2
0061   00      endFor2: STOP
0062                    .END
```



(a) After `scanf("%d", &value)`.



(b) After `printBar(value)`.

Call-by-value with local variables

- To get the actual parameter in the caller, generate load with stack-relative addressing (s)
- To get the formal parameter in the callee, generate load with stack-relative addressing (s)

High-Order Language

```
#include <stdio.h>

void printBar(int n) {
   int k;
   for (k = 1; k <= n; k++) {
      printf("*");
   }
   printf("\n");
}

int main() {
   int numPts;
   int value;
   int j;
   scanf("%d", &numPts);
   for (j = 1; j <= numPts; j++) {
      scanf("%d", &value);
      printBar(value);
   }
   return 0;
}
```

Figure 6.23 (continued)

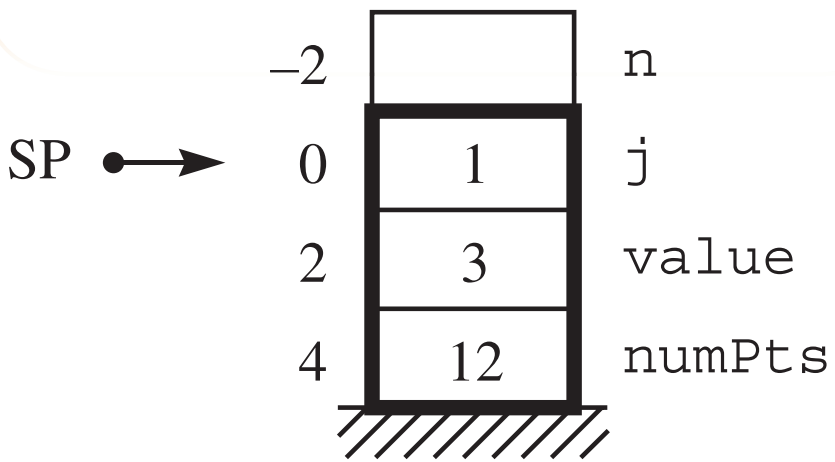
Assembly Language

```
0000  12002E           BR      main
                ;
                ;******* void printBar(int n)
                n:      .EQUATE 4           ;formal parameter #2d
                k:      .EQUATE 0           ;local variable #2d
0003  580002 printBar:SUBSP   2,i           ;push #k
0006  C00001           LDWA    1,i          ;for (k = 1
0009  E30000           STWA    k,s
000C  A30004 for1:     CPWA    n,s          ;k <= n
000F  1E0024           BRGT    endFor1
0012  D0002A           LDBA    '*',i        ;printf("*")
0015  F1FC16           STBA    charOut,d
0018  C30000           LDWA    k,s          ;k++)
001B  600001           ADDA    1,i
001E  E30000           STWA    k,s
0021  12000C           BR      for1
0024  D0000A endFor1:  LDBA    '\n',i       ;printf("\n")
0027  F1FC16           STBA    charOut,d
002A  500002           ADDSP   2,i          ;pop #k
002D  01               RET
```

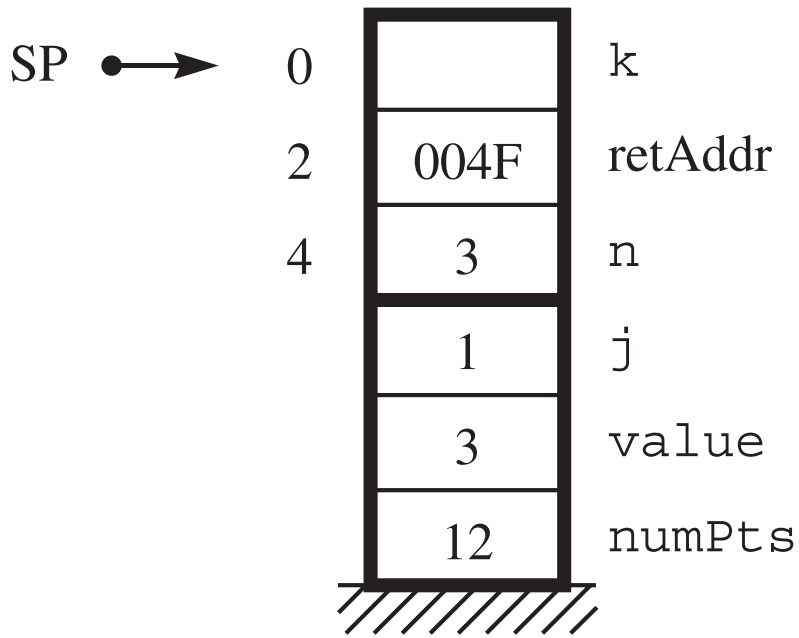
Figure 6.23 (continued)

Assembly Language

```
                ;******* main()
                numPts:  .EQUATE 4          ;local variable #2d
                value:   .EQUATE 2          ;local variable #2d
                j:       .EQUATE 0          ;local variable #2d
002E  580006    main:    SUBSP   6,i        ;push #numPts #value #j
0031  330004             DECI    numPts,s   ;scanf("%d", &numPts)
0034  C00001             LDWA    1,i        ;for (j = 1
0037  E30000             STWA    j,s
003A  A30004    for2:    CPWA    numPts,s   ;j <= numPts
003D  1E005E             BRGT    endFor2
0040  330002             DECI    value,s    ;scanf("%d", &value)
0043  C30002             LDWA    value,s    ;move value
0046  E3FFFE             STWA    -2,s
0049  580002             SUBSP   2,i        ;push #n
004C  240003             CALL    printBar   ;printBar(value)
004F  500002             ADDSP   2,i        ;pop #n
0052  C30000             LDWA    j,s        ;j++)
0055  600001             ADDA    1,i
0058  E30000             STWA    j,s
005B  12003A             BR      for2
005E  500006    endFor2: ADDSP   6,i        ;pop #j #value #numPts
0061  00                 STOP
0062                     .END
```



(a) After `scanf("%d", &value)`.



(b) After `printBar(value)`.

To call a non-`void` function in C

- Push storage for the return value
- Push the actual parameters
- Push the return address
- Push storage for the local variables

High-Order Language

```
#include <stdio.h>

int binCoeff(int n, int k) {
   int y1, y2;
   if ((k == 0) || (n == k)) {
      return 1;
   }
   else {
      y1 = binCoeff(n - 1, k); // ra2
      y2 = binCoeff(n - 1, k - 1); // ra3
      return y1 + y2;
   }
}

int main() {
   printf("binCoeff(3, 1) = %d\n", binCoeff(3, 1)); // ra1
   return 0;
}
```

Figure 6.25 (continued)

Assembly Language

```
0000  12006B           BR      main
                ;
                ;******* int binomCoeff(int n, int k)
                retVal: .EQUATE 10          ;return value #2d
                n:      .EQUATE 8           ;formal parameter #2d
                k:      .EQUATE 6           ;formal parameter #2d
                y1:     .EQUATE 2           ;local variable #2d
                y2:     .EQUATE 0           ;local variable #2d
0003  580004 binCoeff:SUBSP   4,i           ;push #y1 #y2
0006  C30006 if:      LDWA    k,s           ;if ((k == 0)
0009  180015          BREQ    then
000C  C30008          LDWA    n,s           ;|| (n == k))
000F  A30006          CPWA    k,s
0012  1A001F          BRNE    else
0015  C00001 then:    LDWA    1,i           ;return 1
0018  E3000A          STWA    retVal,s
001B  500004          ADDSP   4,i           ;pop #y2 #y1
001E  01              RET
```

Figure 6.25 (continued)

```
001F  C30008 else:   LDWA   n,s          ;move n - 1
0022  700001         SUBA   1,i
0025  E3FFFC         STWA   -4,s
0028  C30006         LDWA   k,s          ;move k
002B  E3FFFA         STWA   -6,s
002E  580006         SUBSP  6,i          ;push #retVal #n #k
0031  240003         CALL   binCoeff     ;binCoeff(n - 1, k)
0034  500006 ra2:    ADDSP  6,i          ;pop #k #n #retVal
0037  C3FFFE         LDWA   -2,s         ;y1 = binomCoeff(n - 1, k)
003A  E30002         STWA   y1,s
003D  C30008         LDWA   n,s          ;move n - 1
0040  700001         SUBA   1,i
0043  E3FFFC         STWA   -4,s
0046  C30006         LDWA   k,s          ;move k - 1
0049  700001         SUBA   1,i
004C  E3FFFA         STWA   -6,s
004F  580006         SUBSP  6,i          ;push #retVal #n #k
0052  240003         CALL   binCoeff     ;binomCoeff(n - 1, k - 1)
0055  500006 ra3:    ADDSP  6,i          ;pop #k #n #retVal
0058  C3FFFE         LDWA   -2,s         ;y2 = binomCoeff(n - 1, k - 1)
005B  E30000         STWA   y2,s
005E  C30002         LDWA   y1,s         ;return y1 + y2
0061  630000         ADDA   y2,s
0064  E3000A         STWA   retVal,s
0067  500004 endIf:  ADDSP  4,i          ;pop #y2 #y1
006A  01             RET
```

Assembly Language

```
                ;******* main()
006B  49008D main:    STRO    msg,d       ;printf("binCoeff(3, 1) = %d\n",
006E  C00003          LDWA    3,i         ;move 3
0071  E3FFFC          STWA    -4,s
0074  C00001          LDWA    1,i         ;move 1
0077  E3FFFA          STWA    -6,s
007A  580006          SUBSP   6,i         ;push #retVal #n #k
007D  240003          CALL    binCoeff    ;binCoeff(3, 1)
0080  500006 ra1:     ADDSP   6,i         ;pop #k #n #retVal
0083  3BFFFE          DECO    -2,s
0086  D0000A          LDBA    '\n',i
0089  F1FC16          STBA    charOut,d
008C  00              STOP
008D  62696E msg:     .ASCII  "binCoeff(3, 1) = \x00"
      436F65
      666628
      332C20
      312920
      3D2000
009F                  .END
```

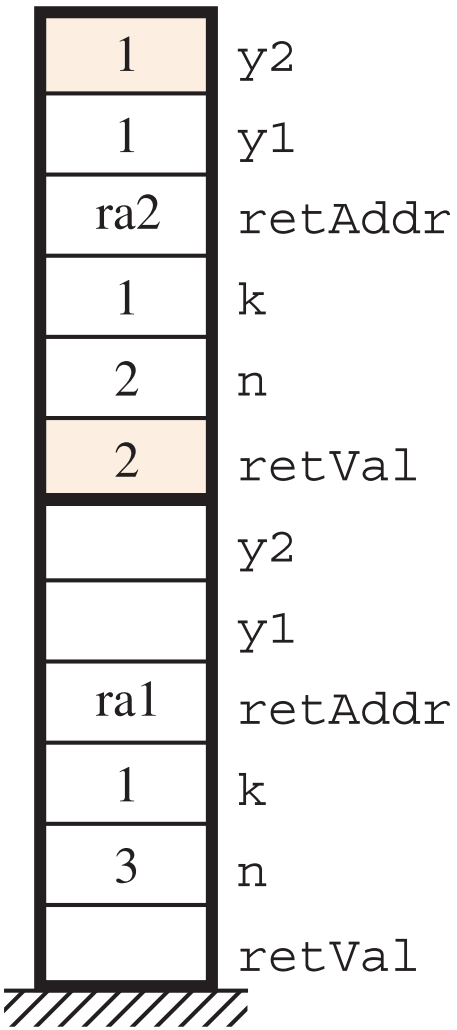


Figure 2.29(g)

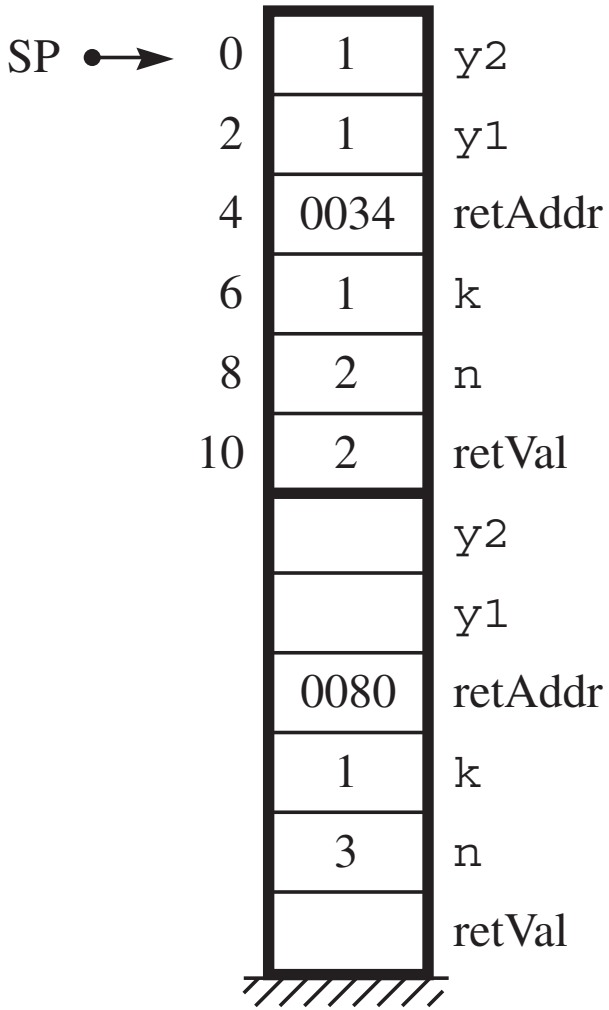


Figure 6.26

Stack-relative deferred addressing

- Oprnd = Mem[Mem[SP + OprndSpec]]
- Asmb5 letters: `sf`

Call-by-reference with global variables

- To get the actual parameter in the caller, generate load with immediate addressing (`i`)
- To get the formal parameter `x` in the callee, generate load with stack-relative addressing (`s`)
- To get the dereferenced formal parameter `*x` in the callee, generate load with stack-relative deferred addressing (`sf`)

High-Order Language

```
#include <stdio.h>

int a, b;

void swap(int *r, int *s) {
   int temp;
   temp = *r;
   *r = *s;
   *s = temp;
}

void order(int *x, int *y) {
   if (*x > *y) {
      swap(x, y);
   }  // ra2
}
```

Figure 6.27 (continued)

High-Order Language

```
int main() {
   printf("Enter an integer: ");
   scanf("%d", &a);
   printf("Enter an integer: ");
   scanf("%d", &b);
   order (&a, &b);
   printf("Ordered they are: %d, %d\n", a, b); // ra1
   return 0;
}
```

Figure 6.27 (continued)

Assembly Language

```
0000  12003F          BR      main
0003  0000    a:      .BLOCK  2           ;global variable #2d
0005  0000    b:      .BLOCK  2           ;global variable #2d
              ;
              ;******* void swap(int *r, int *s)
              r:      .EQUATE 6           ;formal parameter #2h
              s:      .EQUATE 4           ;formal parameter #2h
              temp:   .EQUATE 0           ;local variable #2d
0007  580002  swap:   SUBSP   2,i         ;push #temp
000A  C40006          LDWA    r,sf        ;temp = *r
000D  E30000          STWA    temp,s
0010  C40004          LDWA    s,sf        ;*r = *s
0013  E40006          STWA    r,sf
0016  C30000          LDWA    temp,s      ;*s = temp
0019  E40004          STWA    s,sf
001C  500002          ADDSP   2,i         ;pop #temp
001F  01              RET
```

Figure 6.27 (continued)

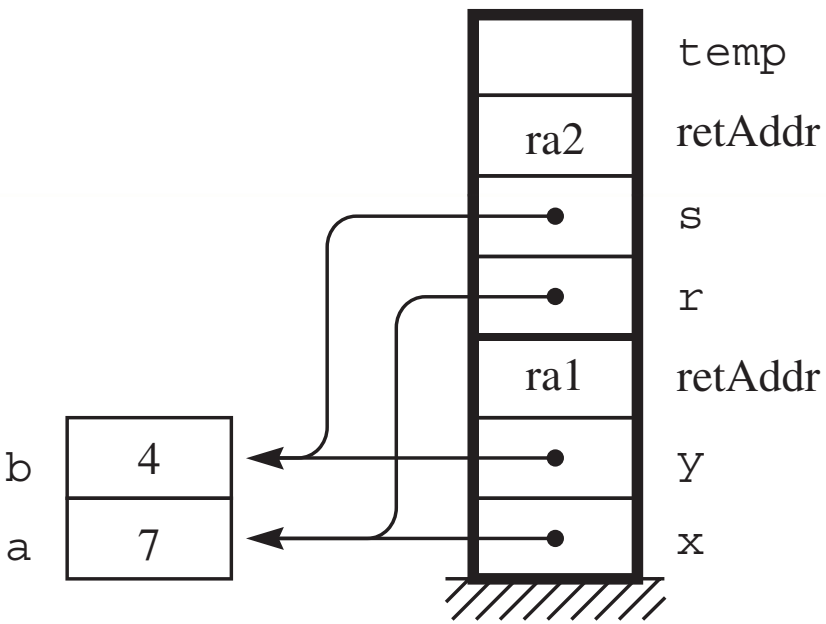
Assembly Language

```
                ;******* void order(int *x, int *y)
                x:      .EQUATE 4           ;formal parameter #2h
                y:      .EQUATE 2           ;formal parameter #2h
0020  C40004    order:  LDWA    x,sf        ;if (*x > *y)
0023  A40002            CPWA    y,sf
0026  14003E            BRLE    endIf
0029  C30004            LDWA    x,s         ;move x
002C  E3FFFE            STWA    -2,s
002F  C30002            LDWA    y,s         ;move y
0032  E3FFFC            STWA    -4,s
0035  580004            SUBSP   4,i         ;push #r #s
0038  240007            CALL    swap        ;swap(x, y)
003B  500004            ADDSP   4,i         ;pop #s #r
003E  01        endIf:  RET
```

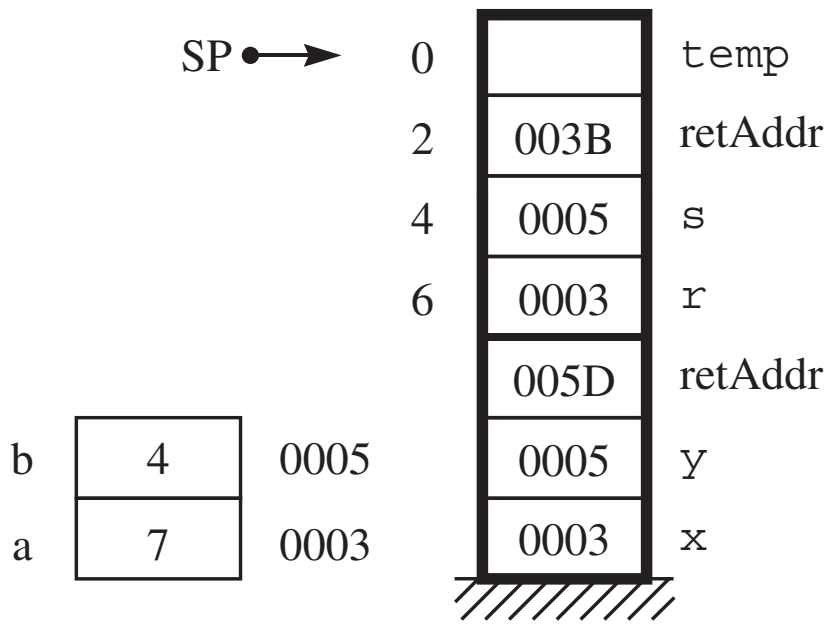
Figure 6.27 (continued)

Assembly Language

```
                ;******* main()
003F  490073 main:    STRO    msg1,d      ;printf("Enter an integer: ")
0042  310003          DECI    a,d         ;scanf("%d", &a)
0045  490073          STRO    msg1,d      ;printf("Enter an integer: ")
0048  310005          DECI    b,d         ;scanf("%d", &b)
004B  C00003          LDWA    a,i         ;move &a
004E  E3FFFE          STWA    -2,s
0051  C00005          LDWA    b,i         ;move &b
0054  E3FFFC          STWA    -4,s
0057  580004          SUBSP   4,i         ;push #x #y
005A  240020          CALL    order       ;order(&a, &b)
005D  500004 ra1:     ADDSP   4,i         ;pop #y #x
0060  490086          STRO    msg2,d      ;printf("Ordered they are: %d, %d\n"
0063  390003          DECO    a,d         ;, a
0066  490099          STRO    msg3,d
0069  390005          DECO    b,d         ;, b)
006C  D0000A          LDBA    '\n',i
006F  F1FC16          STBA    charOut,d
0072  00              STOP
0073  456E74 msg1:    .ASCII  "Enter an integer: \x00"
      ...
0086  4F7264 msg2:    .ASCII  "Ordered they are: \x00"
      ...
0099  2C2000 msg3:    .ASCII  ", \x00"
009C                  .END
```



(a) The run-time stack at Level HOL6.



(b) The run-time stack at Level Asmb5.

The move-SP-to-accumulator instruction

- Instruction specifier: 0000 0011
- Mnemonic: `MOVSPA`
- Accumulator gets the stack pointer

$$A \leftarrow SP$$

Call-by-reference with local variables

- To get the actual parameter in the caller, generate the unary `MOVSPA` followed by `ADDA` with immediate addressing (`i`)
- To get the formal parameter `x` in the callee, generate load with stack-relative addressing (`s`)
- To get the dereferenced formal parameter `*x` in the callee, generate load with stack-relative deferred addressing (`sf`)

High-Order Language

```
#include <stdio.h>

void rect(int *p, int w, int h) {
   *p = (w + h) * 2;
}

int main() {
   int perim, width, height;
   printf("Enter width: ");
   scanf("%d", &width);
   printf("Enter height: ");
   scanf("%d", &height);
   rect(&perim, width, height);
   // ra1
   printf("Perimeter = %d\n", perim);
   return 0;
}
```

Figure 6.29 (continued)

Assembly Language

```
0000  12000E          BR      main
                ;
                ;******* void rect(int *p, int w, int h)
                p:      .EQUATE 6           ;formal parameter #2h
                w:      .EQUATE 4           ;formal parameter #2d
                h:      .EQUATE 2           ;formal parameter #2d
0003  C30004  rect:   LDWA    w,s           ;*p = (w + h) * 2
0006  630002          ADDA    h,s
0009  0A              ASLA
000A  E40006          STWA    p,sf
000D  01              RET
```

Assembly Language

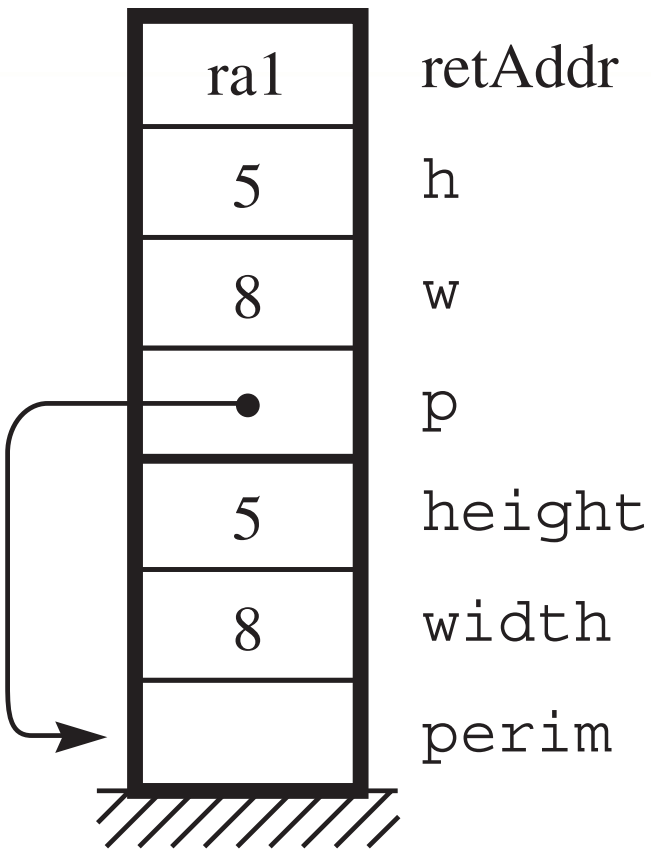
```
                ;******* main()
                perim:   .EQUATE 4           ;local variable #2d
                width:   .EQUATE 2           ;local variable #2d
                height:  .EQUATE 0           ;local variable #2d
000E  580006    main:    SUBSP   6,i         ;push #perim #width #height
0011  490049             STRO    msg1,d      ;printf("Enter width: ")
0014  330002             DECI    width,s     ;scanf("%d", &width)
0017  490057             STRO    msg2,d      ;printf("Enter height: ")
001A  330000             DECI    height,s    ;scanf("%d", &height)
001D  03                 MOVSPA              ;move &perim
001E  600004             ADDA    perim,i
0021  E3FFFE             STWA    -2,s
0024  C30002             LDWA    width,s     ;move width
0027  E3FFFC             STWA    -4,s
002A  C30000             LDWA    height,s    ;move height
002D  E3FFFA             STWA    -6,s
0030  580006             SUBSP   6,i         ;push #p #w #h
0033  240003             CALL    rect        ;rect(&perim, width, height)
0036  500006    ra1:     ADDSP   6,i         ;pop #h #w #p
0039  490066             STRO    msg3,d      ;printf("Perimeter = %d\n", perim);
003C  3B0004             DECO    perim,s
003F  D0000A             LDBA    '\n',i
0042  F1FC16             STBA    charOut,d
0045  500006             ADDSP   6,i         ;pop #height #width #perim
0048  00                 STOP
```

Figure 6.29 (continued)

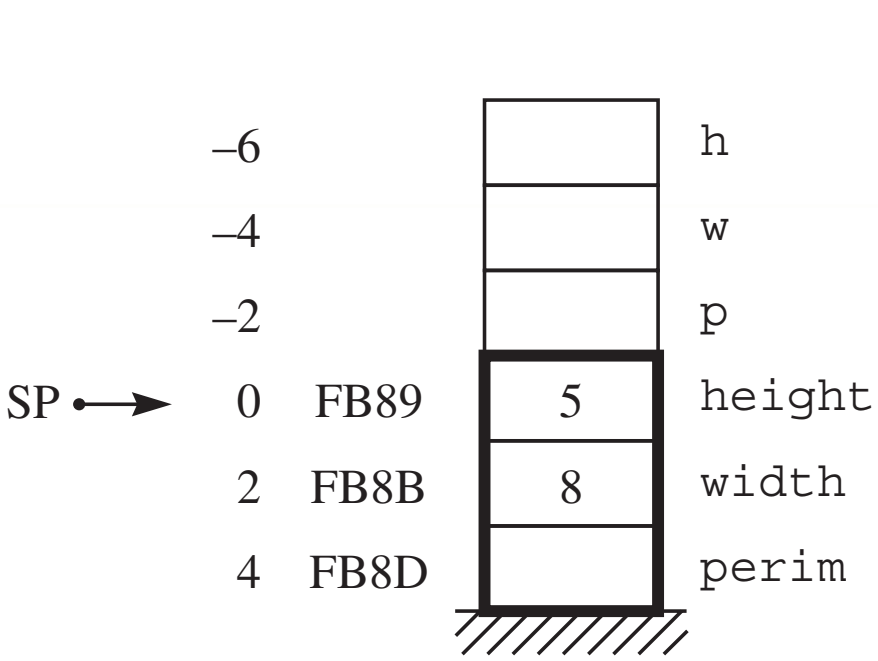
Assembly Language

```
0049   456E74 msg1:    .ASCII   "Enter width: \x00"
       657220
       776964
       74683A
       2000
0057   456E74 msg2:    .ASCII   "Enter height: \x00"
       657220
       686569
       676874
       3A2000
0066   506572 msg3:    .ASCII   "Perimeter = \x00"
       696D65
       746572
       203D20
       00
0073                   .END
```

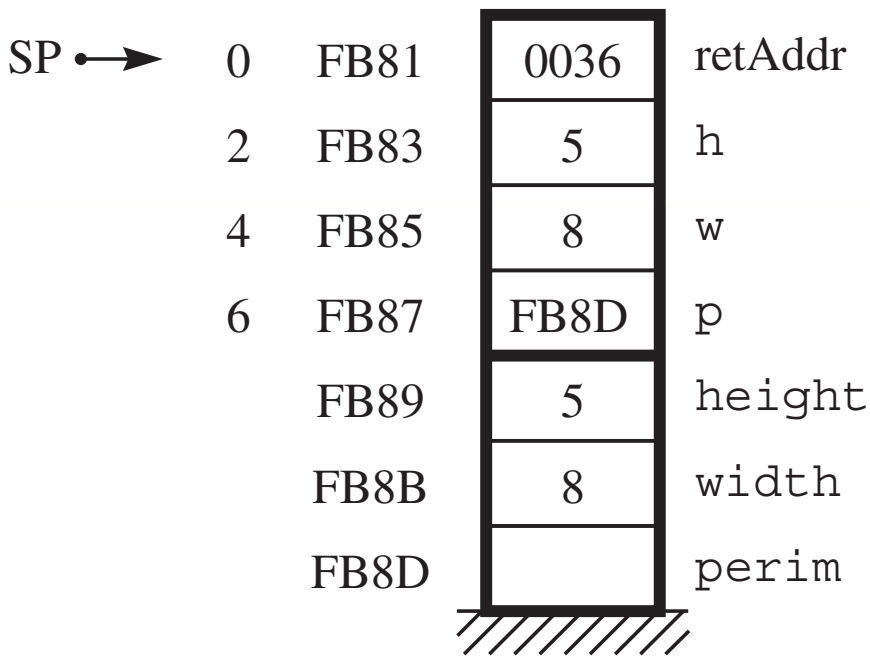
Stack at level HOL6



Stack at level Asmb5



(a) Before the procedure call.



(b) After the procedure call.

Boolean types

```
true:   .EQUATE 1

false:  .EQUATE 0
```

High-Order Language

```
#include <stdio.h>
#include <stdbool.h>

const int LOWER = 21;
const int UPPER = 65;

bool inRange(int a) {
   if ((LOWER <= a) && (a <= UPPER)) {
      return true;
   }
   else {
      return false;
   }
}

int main() {
   int age;
   scanf("%d", &age);
   if (inRange(age)) {
      printf("Qualified\n");
   }
   else {
      printf("Unqualified\n");
   }
   return 0;
}
```

Figure 6.32 (continued)

Assembly Language

```
0000  120023           BR      main
                true:  .EQUATE 1
                false: .EQUATE 0
                ;
                LOWER: .EQUATE 21          ;const int
                UPPER: .EQUATE 65          ;const int
                ;
                ;******* bool inRange(int a)
                retVal: .EQUATE 4          ;returned value #2d
                a:      .EQUATE 2          ;formal parameter #2d
0003  C00015 inRange: LDWA    LOWER,i      ;if ((LOWER <= a)
0006  A30002 if:      CPWA    a,s
0009  1E001C          BRGT    else
000C  C30002          LDWA    a,s          ;&& (a <= UPPER))
000F  A00041          CPWA    UPPER,i
0012  1E001C          BRGT    else
0015  C00001 then:    LDWA    true,i       ;return true
0018  E30004          STWA    retVal,s
001B  01              RET
001C  C00000 else:    LDWA    false,i      ;return false
001F  E30004          STWA    retVal,s
0022  01              RET
```

Figure 6.32 (continued)

Assembly Language

```
                ;******* main()
                age:    .EQUATE 0           ;local variable #2d
0023  580002    main:   SUBSP   2,i         ;push #age
0026  330000            DECI    age,s       ;scanf("%d", &age)
0029  C30000            LDWA    age,s       ;move age
002C  E3FFFC            STWA    -4,s
002F  580004            SUBSP   4,i         ;push #retVal #a
0032  240003            CALL    inRange     ;inRange(age)
0035  500004            ADDSP   4,i         ;pop #a #retVal
0038  C3FFFE            LDWA    -2,s        ;if (inRange(age))
003B  180044            BREQ    else2
003E  49004B    then2:  STRO    msg1,d      ;printf("Qualified\n")
0041  120047            BR      endif2
0044  490056    else2:  STRO    msg2,d      ;printf("Unqualified\n");
0047  500002    endif2: ADDSP   2,i         ;pop #age
004A  00                STOP
004B  517561    msg1:   .ASCII  "Qualified\n\x00"
      ...
0056  556E71    msg2:   .ASCII  "Unqualified\n\x00"
      ...
0063                    .END
```

Figure 6.33

Addressing Mode	aaa	Letters	Operand
Immediate	000	i	OprndSpec
Direct	001	d	Mem[OprndSpec]
Indirect	010	n	Mem[Mem[OprndSpec]]
Stack-relative	011	s	Mem[SP + OprndSpec]
Stack-relative deferred	100	sf	Mem[Mem[SP + OprndSpec]]
Indexed	101	x	Mem[OprndSpec + X]
Stack-indexed	110	sx	Mem[SP + OprndSpec + X]
Stack-deferred indexed	111	sfx	Mem[Mem[SP + OprndSpec] + X]

Indexed addressing

- Oprnd = Mem[OprndSpec + X]
- Asmb5 letter: `x`

Global arrays

- Allocate number of bytes with `.BLOCK` *tot*, where *tot* is the total number of bytes occupied by the array
- To get element `v[i]`, load `i` into index register, multiply `i` by number of bytes per cell, (`ASLX` in the case of an an array of integers), and use indexed addressing (`x`)

High-Order Language

```
#include <stdio.h>

int vector[4];
int j;

int main() {
   for (j = 0; j < 4; j++) {
      scanf("%d", &vector[j]);
   }
   for (j = 3; j >= 0; j--) {
      printf("%d %d\n", j, vector[j]);
   }
   return 0;
}
```

Figure 6.34 (continued)

Assembly Language

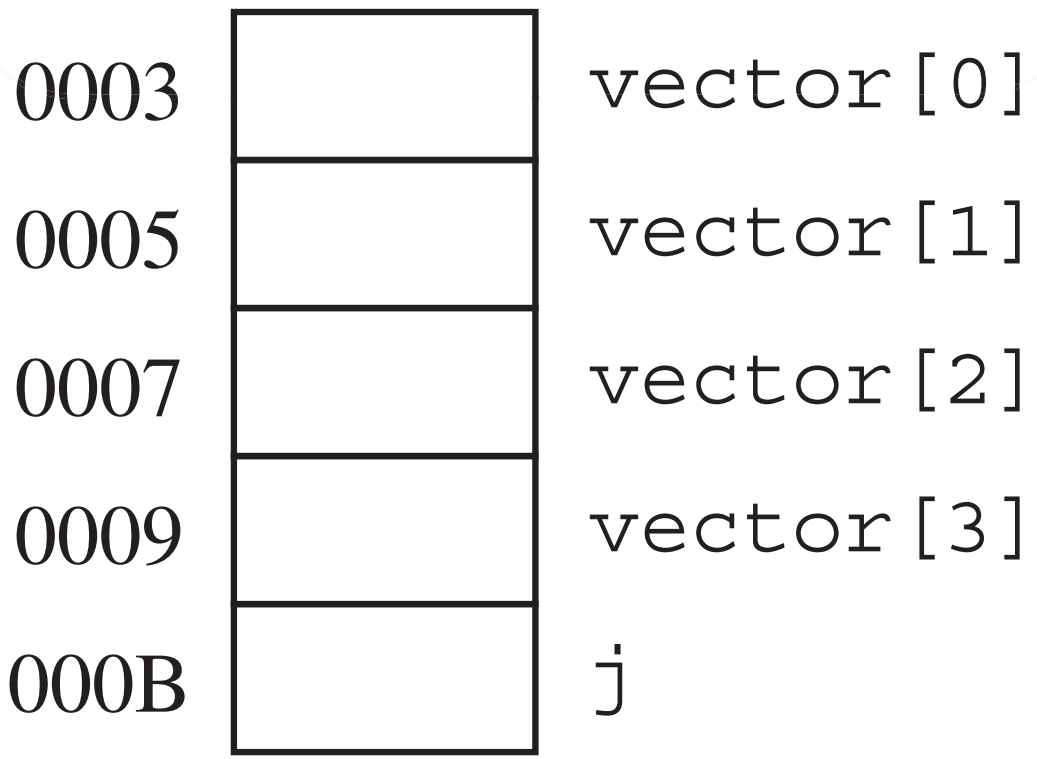
```
0000   12000D            BR      main
0003   000000 vector:    .BLOCK  8           ;global variable #2d4a
       000000
       0000
000B   0000   j:         .BLOCK  2           ;global variable #2d
              ;
              ;******* main()
000D   C80000 main:      LDWX    0,i         ;for (j = 0
0010   E9000B            STWX    j,d
0013   A80004 for1:      CPWX    4,i         ;j < 4
0016   1C0029            BRGE    endFor1
0019   0B                ASLX                ;two bytes per integer
001A   350003            DECI    vector,x    ;scanf("%d", &vector[j])
001D   C9000B            LDWX    j,d         ;j++)
0020   680001            ADDX    1,i
0023   E9000B            STWX    j,d
0026   120013            BR      for1
```

Figure 6.34 (continued)

Assembly Language

```
0029  C80003 endFor1: LDWX   3,i          ;for (j = 3
002C  E9000B          STWX   j,d
002F  A80000 for2:    CPWX   0,i          ;j >= 0
0032  160054          BRLT   endFor2
0035  39000B          DECO   j,d          ;printf("%d %d\n", j, vector[j])
0038  D00020          LDBA   ' ',i
003B  F1FC16          STBA   charOut,d
003E  0B              ASLX                ;two bytes per integer
003F  3D0003          DECO   vector,x
0042  D0000A          LDBA   '\n',i
0045  F1FC16          STBA   charOut,d
0048  C9000B          LDWX   j,d          ;j--)
004B  780001          SUBX   1,i
004E  E9000B          STWX   j,d
0051  12002F          BR     for2
0054  00      endFor2: STOP
0055                  .END
```

Figure 6.35



Stack-indexed addressing

- Oprnd = Mem[SP + OprndSpec + X]
- Asmb5 letters: `sx`

Local arrays

- Allocate number of bytes with `SUBSP` *tot*, with immediate addressing, where *tot* is the total number of bytes occupied by the array
- To get element `v[i]`, load `i` into index register, multiply `i` by number of bytes per cell, (`ASLX` in the case of an an array of integers), and use stack-indexed addressing (`sx`)

High-Order Language

```
#include <stdio.h>

int main() {
   int vector[4];
   int j;
   for (j = 0; j < 4; j++) {
      scanf("%d", &vector[j]);
   }
   for (j = 3; j >= 0; j--) {
      printf("%d %d\n", j, vector[j]);
   }
   return 0;
}
```

Figure 6.36 (continued)

Assembly Language

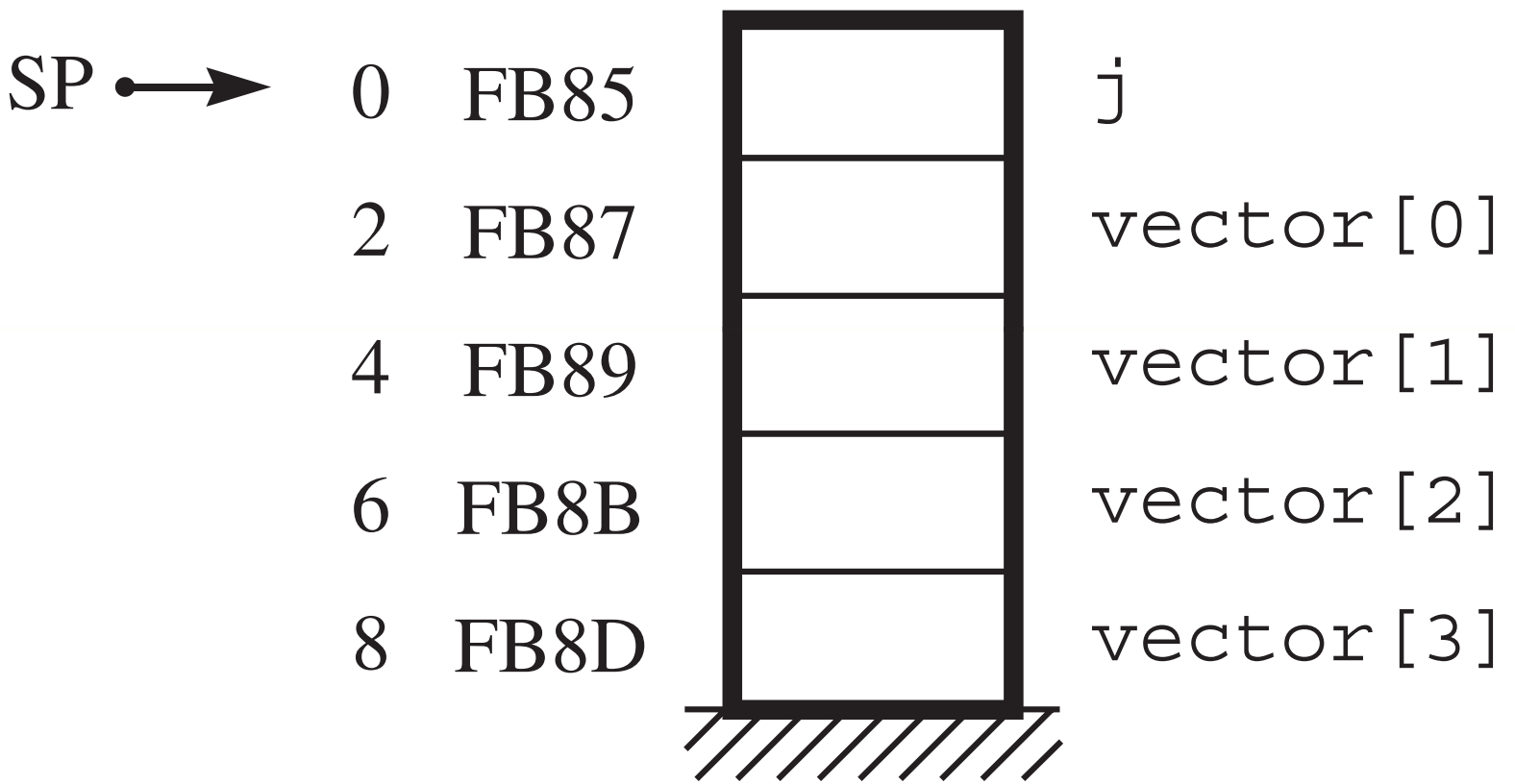
```
0000   120003           BR      main
               ;
               ;******* main ()
               vector:  .EQUATE 2           ;local variable #2d4a
               j:       .EQUATE 0           ;local variable #2d
0003   58000A main:     SUBSP   10,i        ;push #vector #j
0006   C80000           LDWX    0,i         ;for (j = 0
0009   EB0000           STWX    j,s
000C   A80004 for1:     CPWX    4,i         ;j < 4
000F   1C0022           BRGE    endFor1
0012   0B               ASLX                ;two bytes per integer
0013   360002           DECI    vector,sx   ;scanf("%d", &vector[j])
0016   CB0000           LDWX    j,s         ;j++)
0019   680001           ADDX    1,i
001C   EB0000           STWX    j,s
001F   12000C           BR      for1
```

Figure 6.36 (continued)

Assembly Language

```
0022  C80003 endFor1: LDWX    3,i          ;for (j = 3
0025  EB0000          STWX    j,s
0028  A80000 for2:    CPWX    0,i          ;j >= 0
002B  16004D          BRLT    endFor2
002E  3B0000          DECO    j,s          ;printf("%d %d\n", j, vector[j])
0031  D00020          LDBA    ' ',i
0034  F1FC16          STBA    charOut,d
0037  0B              ASLX                 ;two bytes per integer
0038  3E0002          DECO    vector,sx
003B  D0000A          LDBA    '\n',i
003E  F1FC16          STBA    charOut,d
0041  CB0000          LDWX    j,s          ;j--)
0044  780001          SUBX    1,i
0047  EB0000          STWX    j,s
004A  120028          BR      for2
004D  50000A endFor2: ADDSP   10,i         ;pop #j #vector
0050  00              STOP
0051                  .END
```

Figure 6.37



Stack-deferred indexed addressing

- Oprnd = Mem [Mem[SP + OprndSpec] + X]
- Asmb5 letters: `sfx`

Passing a local array as a parameter

- To get the actual parameter in the caller, generate the unary `MOVSPA` followed by `ADDA` with immediate addressing (`i`)
- To get element `v[i]` in the callee, load `i` into index register, multiply `i` by number of bytes per cell, (`ASLX` in the case of an an array of integers), and use stack-deferred indexed addressing (`sfx`)

High-Order Language

```
#include <stdio.h>

void getVect(int v[], int *n) {
   int j;
   scanf("%d", n);
   for (j = 0; j < *n; j++) {
      scanf("%d", &v[j]);
   }
}

void putVect(int v[], int n) {
   int j;
   for (j = 0; j < n; j++) {
      printf("%d ", v[j]);
   }
   printf("\n");
}

int main() {
   int vector[8];
   int numItms;
   getVect(vector, &numItms);
   putVect(vector, numItms);
   return 0;
}
```

Figure 6.38 (continued)

Assembly Language

```
0000  120058           BR      main
                ;
                ;******* getVect(int v[], int *n)
                v:      .EQUATE 6           ;formal parameter #2h
                n:      .EQUATE 4           ;formal parameter #2h
                j:      .EQUATE 0           ;local variable #2d
0003  580002 getVect: SUBSP   2,i         ;push #j
0006  340004           DECI    n,sf        ;scanf("%d", n)
0009  C80000           LDWX    0,i         ;for (j = 0
000C  EB0000           STWX    j,s
000F  AC0004 for1:     CPWX    n,sf        ;j < *n
0012  1C0025           BRGE    endFor1
0015  0B               ASLX                ;two bytes per integer
0016  370006           DECI    v,sfx       ;scanf("%d", &v[j])
0019  CB0000           LDWX    j,s         ;j++)
001C  680001           ADDX    1,i
001F  EB0000           STWX    j,s
0022  12000F           BR      for1
0025  500002 endFor1: ADDSP   2,i         ;pop #j
0028  01               RET
```

Figure 6.38 (continued)

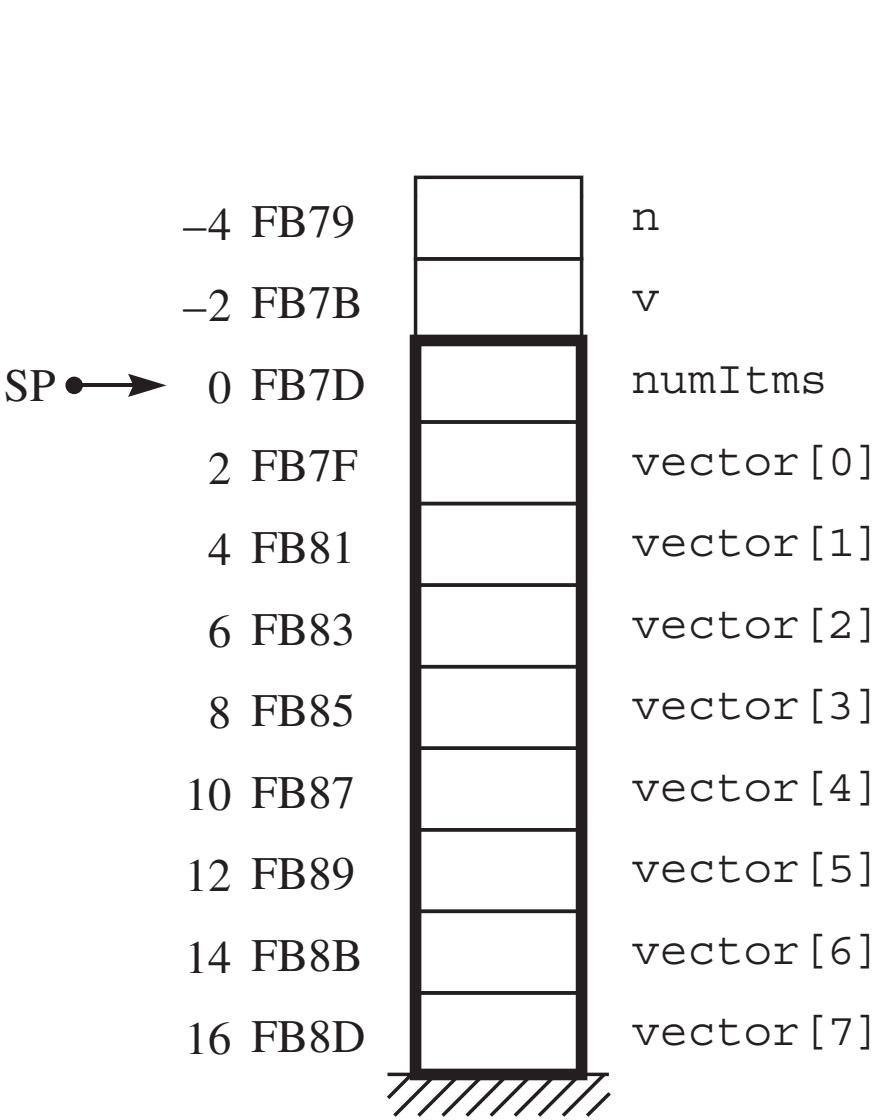
Assembly Language

```
                ;******* putVect(int v[], int n)
                v2:     .EQUATE 6           ;formal parameter #2h
                n2:     .EQUATE 4           ;formal parameter #2d
                j2:     .EQUATE 0           ;local variable #2d
0029  580002 putVect: SUBSP   2,i           ;push #j2
002C  C80000          LDWX    0,i           ;for (j = 0
002F  EB0000          STWX    j2,s
0032  AB0004 for2:    CPWX    n2,s          ;j < n
0035  1C004E          BRGE    endFor2
0038  0B              ASLX                  ;two bytes per integer
0039  3F0006          DECO    v2,sfx        ;printf("%d ", v[j])
003C  D00020          LDBA    ' ',i
003F  F1FC16          STBA    charOut,d
0042  CB0000          LDWX    j2,s          ;j++)
0045  680001          ADDX    1,i
0048  EB0000          STWX    j2,s
004B  120032          BR      for2
004E  D0000A endFor2: LDBA    '\n',i        ;printf("\n")
0051  F1FC16          STBA    charOut,d
0054  500002          ADDSP   2,i           ;pop #j2
0057  01              RET
```

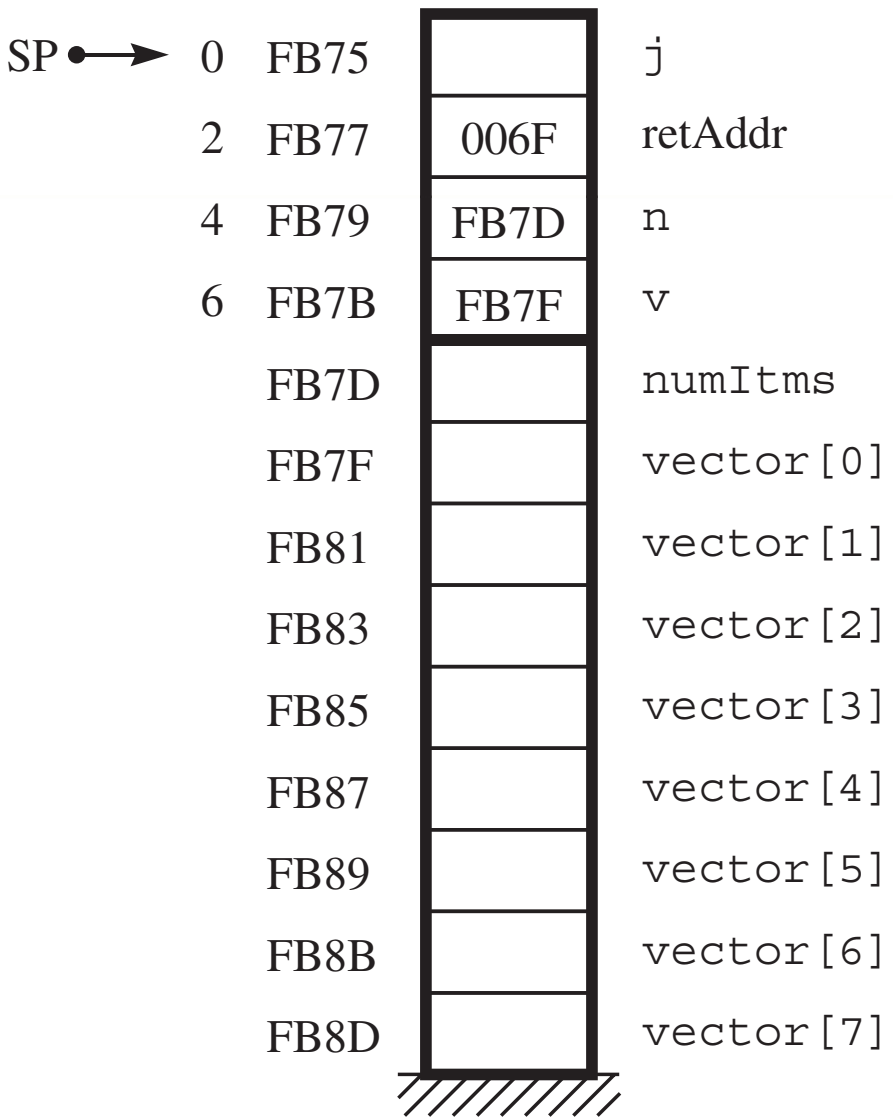
Figure 6.38 (continued)

Assembly Language

```
                ;******* main()
                vector:  .EQUATE 2           ;local variable #2d8a
                numItms: .EQUATE 0           ;local variable #2d
0058  580012  main:    SUBSP   18,i          ;push #vector #numItms
005B  03               MOVSPA                ;move (&)vector
005C  600002           ADDA    vector,i
005F  E3FFFE           STWA    -2,s
0062  03               MOVSPA                ;move &numItms
0063  600000           ADDA    numItms,i
0066  E3FFFC           STWA    -4,s
0069  580004           SUBSP   4,i           ;push #v #n
006C  240003           CALL    getVect       ;getVect(vector, &numItms)
006F  500004           ADDSP   4,i           ;pop #n #v
0072  03               MOVSPA                ;move (&)vector
0073  600002           ADDA    vector,i
0076  E3FFFE           STWA    -2,s
0079  C30000           LDWA    numItms,s     ;move numItms
007C  E3FFFC           STWA    -4,s
007F  580004           SUBSP   4,i           ;push #v2 #n2
0082  240029           CALL    putVect       ;putVect(vector, numItms)
0085  500004           ADDSP   4,i           ;pop #n2 #v2
0088  500012           ADDSP   18,i          ;pop #numItms #vector
008B  00               STOP
008C                   .END
```



(a) Before calling `getVect( )`.



(b) After calling `getVect( )`.

Passing a global array as a parameter

- To get the actual parameter in the caller, generate the `LDWA` with immediate addressing (`i`)
- To get element `v[i]` in the callee, load `i` into index register, multiply `i` by number of bytes per cell, (`ASLX` in the case of an an array of integers), and use stack-deferred indexed addressing (`sfx`)

(No example program)

.ADDRSS

- Every `.ADDRSS` command must be followed by a symbol
- The code generated by `.ADDRSS` is the value of the symbol

The `switch` statement

- Set up a jump table with `.ADDRSS`
- Use `LDWX` to load the index register with the switch value
- Execute `ASLX` once, because an address occupies two bytes
- Execute `BR` with indexed addressing (`x`)

High-Order Language

```
#include <stdio.h>

int main() {
   int guess;
   printf("Pick a number 0..3: ");
   scanf("%d", &guess);
   switch (guess) {
      case 0: printf("Not close\n"); break;
      case 1: printf("Close\n"); break;
      case 2: printf("Right on\n"); break;
      case 3: printf("Too high\n");
   }
   return 0;
}
```

Figure 6.40 (continued)

Assembly Language

```
0000  120003          BR      main
                ;
                ;******* main()
                guess:  .EQUATE 0           ;local variable #2d
0003  580002  main:     SUBSP   2,i         ;push #guess
0006  490034            STRO    msgIn,d     ;printf("Pick a number 0..3: ")
0009  330000            DECI    guess,s     ;scanf("%d", &guess)
000C  CB0000            LDWX    guess,s     ;switch (guess)
000F  0B                ASLX                ;two bytes per address
0010  130013            BR      guessJT,x
0013  001B    guessJT:  .ADDRSS case0
0015  0021              .ADDRSS case1
0017  0027              .ADDRSS case2
0019  002D              .ADDRSS case3
001B  490049  case0:    STRO    msg0,d      ;printf("Not close\n")
001E  120030            BR      endCase     ;break
0021  490054  case1:    STRO    msg1,d      ;printf("Close\n")
0024  120030            BR      endCase     ;break
0027  49005B  case2:    STRO    msg2,d      ;printf("Right on\n")
002A  120030            BR      endCase     ;break
002D  490065  case3:    STRO    msg3,d      ;printf("Too high\n")
0030  500002  endCase:  ADDSP   2,i         ;pop #guess
0033  00                STOP
```

Figure 6.40 (continued)

Assembly Language

```
0034   506963 msgIn:    .ASCII  "Pick a number 0..3: \x00"
       ...
0049   4E6F74 msg0:     .ASCII  "Not close\n\x00"
       ...
0054   436C6F msg1:     .ASCII  "Close\n\x00"
       ...
005B   526967 msg2:     .ASCII  "Right on\n\x00"
       ...
0065   546F6F msg3:     .ASCII  "Too high\n\x00"
       ...
006F                    .END
```

Symbol table

Symbol	Value	Symbol	Value
case0	001B	case1	0021
case2	0027	case3	002D
endCase	0030	guess	0000
guessJT	0013	main	0003
msg0	0049	msg1	0054
msg2	005B	msg3	0065
msgIn	0034		

Intel x86 addressing modes

- Register
- Immediate
- Direct
- Base
- Base + Displacement
- Index + Displacement
- Scaled Index + Displacement
- Based Index
- Based Scaled Index
- Based Index + Displacement
- Based Scaled Index + Displacement
- PC Relative

```
while (letter !='*')
```

```
0003a   0f be 05 00 00
        00 00 00          movsx eax, BYTE PTR _letter
00041   83 f8 2a          cmp   eax, 42 ; 0000002aH
00044   74 62             je    SHORT $LN3@main
```

(a) Translation of the `while` loop test.

```
printBar(value)
```

```
0007c   a1 00 00 00 00 mov    eax, DWORD PTR _value
00081   50             push   eax
00082   e8 00 00 00 00 call   _printBar
00087   83 c4 04        add    esp, 4
```

(b) Translation of the `printBar()` function call.

Figure 6.41 (continued)

```
for (k = 1; k<= n; k++)
```

```
0001e   c7 45 f8 01 00 ; k = 1
        00 00          mov  DWORD PTR _k$[ebp], 1
00025   eb 09          jmp  SHORT $LN3@printBar
$LN2@printBar:         ; k++
00027   8b 45 f8       mov  eax, DWORD PTR _k$[ebp]
0002a   83 c0 01       add  eax, 1
0002d   89 45 f8       mov  DWORD PTR _k$[ebp], eax
$LN3@printBar:         ; k <= n
00030   8b 45 f8       mov  eax, DWORD PTR _k$[ebp]
00033   3b 45 08       cmp  eax, DWORD PTR _n$[ebp]
00036   7f 19          jg   SHORT $LN1@printBar
```

(c) Translation of the `for` loop.

```
for (k = 1; k<= n; k++)
```

```
0004f eb d6                jmp SHORT $LN2@printBar
$LN1@printBar:
```

(d) The branch to the top of the loop.

Immediate addressing

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

Direct addressing

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

Inirect addressing

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

Global pointers

- Allocate storage for the pointer with `.BLOCK 2` because an address occupies two bytes
- To get the pointer, generate `LDWA` with direct addressing (d)
- To get the content of the cell to which the pointer points, generate `LDWA` or `LDBA` with indirect addressing (`n`)

Function `malloc()` calling protocol

- Put number of bytes to be allocated in accumulator
- `CALL malloc`
- The index register will contain a pointer to the allocated bytes

High-Order Language

```
#include <stdio.h>
#include <stdlib.h>

int *a, *b, *c;

int main() {
   a = (int *) malloc(sizeof(int));
   *a = 5;
   b = (int *) malloc(sizeof(int));
   *b = 3;
   c = a;
   a = b;
   *a = 2 + *c;
   printf("*a = %d\n", *a);
   printf("*b = %d\n", *b);
   printf("*c = %d\n", *c);
   return 0;
}
```

Figure 6.42 (continued)

Assembly Language

```
0000  120009          BR      main
0003  0000    a:      .BLOCK  2           ;global variable #2h
0005  0000    b:      .BLOCK  2           ;global variable #2h
0007  0000    c:      .BLOCK  2           ;global variable #2h
              ;
              ;******* main ()
0009  C00002  main:   LDWA    2,i         ;a = (int *) malloc(sizeof(int))
000C  240073          CALL    malloc      ;allocate #2d
000F  E90003          STWX    a,d
0012  C00005          LDWA    5,i         ;*a = 5
0015  E20003          STWA    a,n
0018  C00002          LDWA    2,i         ;b = (int *) malloc(sizeof(int))
001B  240073          CALL    malloc      ;allocate #2d
001E  E90005          STWX    b,d
0021  C00003          LDWA    3,i         ;*b = 3
0024  E20005          STWA    b,n
0027  C10003          LDWA    a,d         ;c = a
002A  E10007          STWA    c,d
002D  C10005          LDWA    b,d         ;a = b
0030  E10003          STWA    a,d
0033  C00002          LDWA    2,i         ;*a = 2 + *c
0036  620007          ADDA    c,n
0039  E20003          STWA    a,n
```

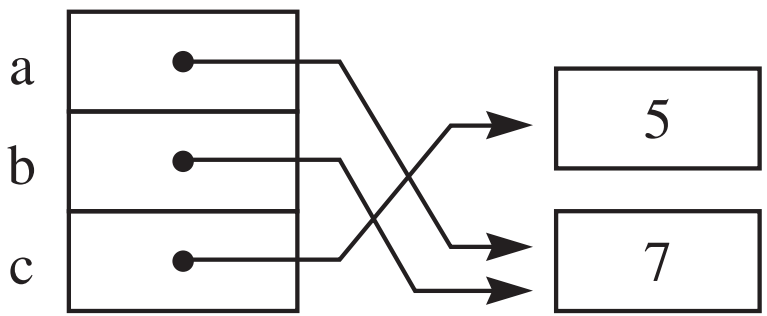
Figure 6.42 (continued)

Assembly Language

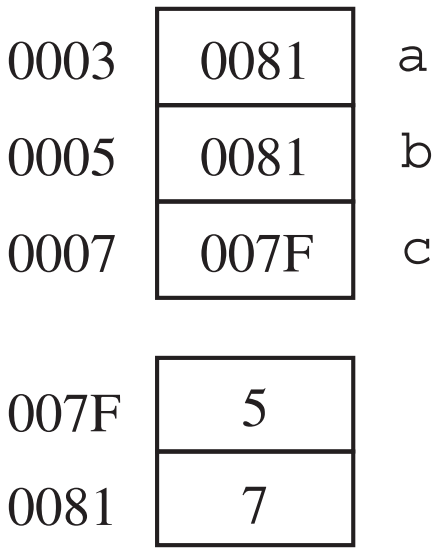
```
003C  490061          STRO    msg0,d      ;printf("*a = %d\n", *a)
003F  3A0003          DECO    a,n
0042  D0000A          LDBA    '\n',i
0045  F1FC16          STBA    charOut,d
0048  490067          STRO    msg1,d      ;printf("*b = %d\n", *b)
004B  3A0005          DECO    b,n
004E  D0000A          LDBA    '\n',i
0051  F1FC16          STBA    charOut,d
0054  49006D          STRO    msg2,d      ;printf("*c = %d\n", *c)
0057  3A0007          DECO    c,n
005A  D0000A          LDBA    '\n',i
005D  F1FC16          STBA    charOut,d
0060  00              STOP
0061  2A6120 msg0:    .ASCII  "*a = \x00"
      3D2000
0067  2A6220 msg1:    .ASCII  "*b = \x00"
      3D2000
006D  2A6320 msg2:    .ASCII  "*c = \x00"
      3D2000
```

Assembly Language

```
                ;******* malloc()
                ;       Precondition: A contains number of bytes
                ;       Postcondition: X contains pointer to bytes
0073  C9007D  malloc:  LDWX    hpPtr,d      ;returned pointer
0076  61007D           ADDA    hpPtr,d      ;allocate from heap
0079  E1007D           STWA    hpPtr,d      ;update hpPtr
007C  01               RET
007D  007F    hpPtr:   .ADDRSS heap         ;address of next free byte
007F  00      heap:    .BLOCK  1            ;first byte in the heap
0080                   .END
```



(a) Global pointers at Level HOL6.



(b) The same global pointers at Level Asmb5.

Local pointers

- Allocate storage for the pointer with `SUBSP` with two bytes for each pointer
- To get the pointer, generate LDWA with stack-relative addressing (`s`)
- To get the content of the cell to which the pointer points, generate `LDWA` with stack-relative deferred addressing (`sf`)

High-Order Language

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int *a, *b, *c;
   a = (int *) malloc(sizeof(int));
   *a = 5;
   b = (int *) malloc(sizeof(int));
   *b = 3;
   c = a;
   a = b;
   *a = 2 + *c;
   printf("*a = %d\n", *a);
   printf("*b = %d\n", *b);
   printf("*c = %d\n", *c);
   return 0;
}
```

Figure 6.44 (continued)

Assembly Language

```
0000   120003           BR      main
                ;
                ;******* main()
                a:      .EQUATE 4           ;local variable #2h
                b:      .EQUATE 2           ;local variable #2h
                c:      .EQUATE 0           ;local variable #2h
0003   580006 main:     SUBSP   6,i         ;push #a #b #c
0006   C00002           LDWA    2,i         ;a = (int *) malloc(sizeof(int))
0009   240073           CALL    malloc      ;allocate #2d
000C   EB0004           STWX    a,s
000F   C00005           LDWA    5,i         ;*a = 5
0012   E40004           STWA    a,sf
0015   C00002           LDWA    2,i         ;b = (int *) malloc(sizeof(int))
0018   240073           CALL    malloc      ;allocate #2d
001B   EB0002           STWX    b,s
001E   C00003           LDWA    3,i         ;*b = 3
0021   E40002           STWA    b,sf
0024   C30004           LDWA    a,s         ;c = a
0027   E30000           STWA    c,s
002A   C30002           LDWA    b,s         ;a = b
002D   E30004           STWA    a,s
0030   C00002           LDWA    2,i         ;*a = 2 + *c
0033   640000           ADDA    c,sf
0036   E40004           STWA    a,sf
```

Figure 6.44 (continued)

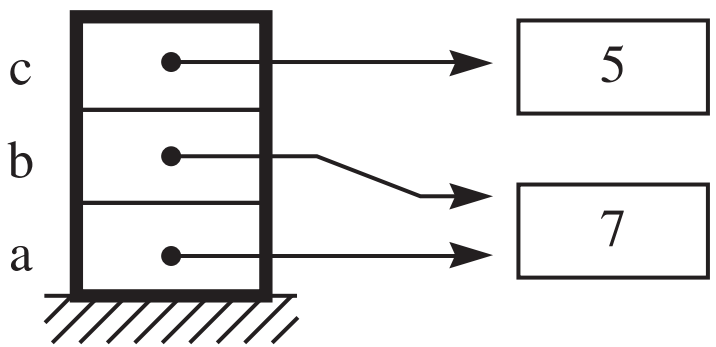
Assembly Language

```
0039  490061           STRO    msg0,d       ;printf("*a = %d\n", *a)
003C  3C0004           DECO    a,sf
003F  D0000A           LDBA    '\n',i
0042  F1FC16           STBA    charOut,d
0045  490067           STRO    msg1,d       ;printf("*b = %d\n", *b)
0048  3C0002           DECO    b,sf
004B  D0000A           LDBA    '\n',i
004E  F1FC16           STBA    charOut,d
0051  49006D           STRO    msg2,d       ;printf("*c = %d\n", *c)
0054  3C0000           DECO    c,sf
0057  D0000A           LDBA    '\n',i
005A  F1FC16           STBA    charOut,d
005D  500006           ADDSP   6,i          ;pop #c #b #a
0060  00               STOP
0061  2A6120 msg0:     .ASCII  "*a = \x00"
      3D2000
0067  2A6220 msg1:     .ASCII  "*b = \x00"
      3D2000
006D  2A6320 msg2:     .ASCII  "*c = \x00"
      3D2000
```

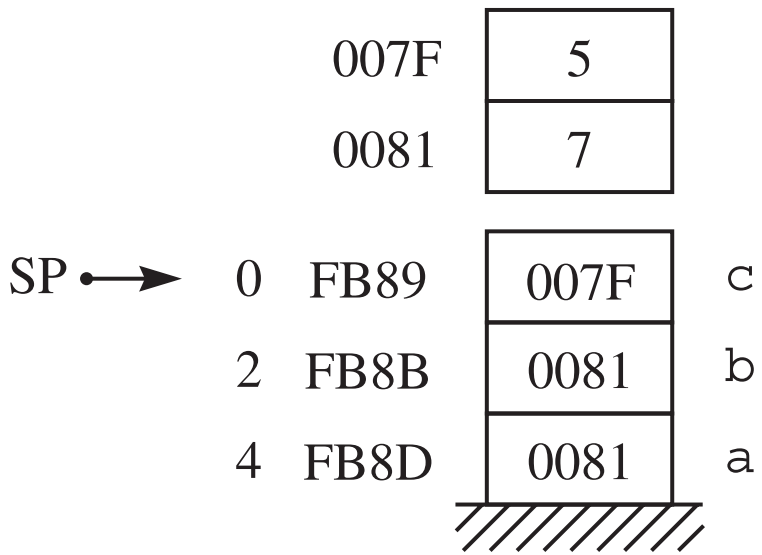
Figure 6.44 (continued)

Assembly Language

```
                ;******* malloc()
                ;       Precondition: A contains number of bytes
                ;       Postcondition: X contains pointer to bytes
0073  C9007D  malloc:  LDWX    hpPtr,d     ;returned pointer
0076  61007D           ADDA    hpPtr,d     ;allocate from heap
0079  E1007D           STWA    hpPtr,d     ;update hpPtr
007C  01               RET
007D  007F    hpPtr:   .ADDRSS heap        ;address of next free byte
007F  00      heap:    .BLOCK  1           ;first byte in the heap
0080                   .END
```



(a) Local pointers at Level HOL6.



(b) The same local pointers at Level Asmb5.

Global structures

- Equate each field of the `struct` to its offset from the first byte of the `struct`
- Allocate storage for `struct` with `.BLOCK` *tot*, where *tot* is the total number of bytes occupied by the structure
- To get a field, generate `LDWX` with immediate addressing (`i`) to put the field into the index register, followed by a load with indexed addressing (`x`)

High-Order Language

```
#include <stdio.h>

struct person {
   char first;
   char last;
   int age;
   char gender;
};
struct person bill;

int main() {
   scanf("%c%c%d %c", &bill.first, &bill.last, &bill.age, &bill.gender);
   printf("Initials: %c%c\n", bill.first, bill.last);
   printf("Age: %d\n", bill.age);
   printf("Gender: ");
   if (bill.gender == 'm') {
      printf("male\n");
   }
   else {
      printf("female\n");
   }
   return 0;
}
```

Figure 6.46 (continued)

Assembly Language

```
0000  120008          BR      main
                first:  .EQUATE 0           ;struct field #1c
                last:   .EQUATE 1           ;struct field #1c
                age:    .EQUATE 2           ;struct field #2d
                gender: .EQUATE 4           ;struct field #1c
0003  000000  bill:   .BLOCK  5           ;globals #first #last #age #gender
      0000
                ;
                ;******* main()
0008  C80000  main:   LDWX    first,i     ;scanf("%c%c%d %c", &bill.first,
000B  D1FC15          LDBA    charIn,d
000E  F50003          STBA    bill,x
0011  C80001          LDWX    last,i      ;&bill.last,
0014  D1FC15          LDBA    charIn,d
0017  F50003          STBA    bill,x
001A  C80002          LDWX    age,i       ;&bill.age,
001D  350003          DECI    bill,x
0020  C80004          LDWX    gender,i    ;&bill.gender)
0023  D1FC15          LDBA    charIn,d
0026  F50003          STBA    bill,x
```

Figure 6.46 (continued)

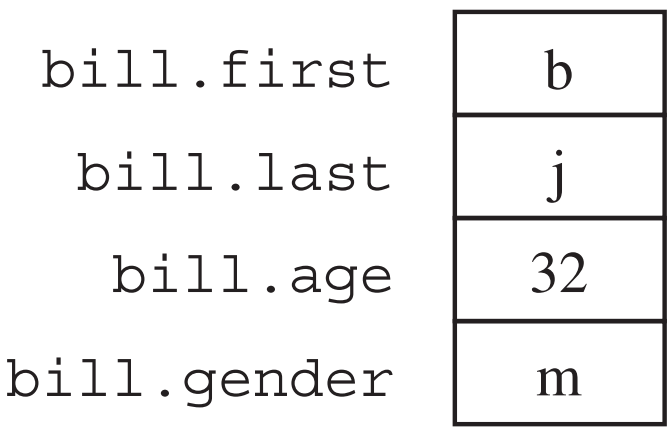
Assembly Language

```
0029  49006C          STRO    msg0,d      ;printf("Initials: %c%c\n",
002C  C80000          LDWX    first,i     ;bill.first,
002F  D50003          LDBA    bill,x
0032  F1FC16          STBA    charOut,d
0035  C80001          LDWX    last,i      ;bill.last)
0038  D50003          LDBA    bill,x
003B  F1FC16          STBA    charOut,d
003E  D0000A          LDBA    '\n',i
0041  F1FC16          STBA    charOut,d
0044  490077          STRO    msg1,d      ;printf("Age:  %d\n",
0047  C80002          LDWX    age,i       ;bill.age)
004A  3D0003          DECO    bill,x
004D  D0000A          LDBA    '\n',i
0050  F1FC16          STBA    charOut,d
0053  49007D          STRO    msg2,d      ;printf("Gender: ")
0056  C80004          LDWX    gender,i    ;if (bill.gender == 'm')
0059  D50003          LDBA    bill,x
005C  B0006D          CPBA    'm',i
005F  1A0068          BRNE    else
0062  490086          STRO    msg3,d      ;printf("male\n")
0065  12006B          BR      endIf
0068  49008C  else:   STRO    msg4,d      ;printf("female\n")
006B  00      endIf:  STOP
```

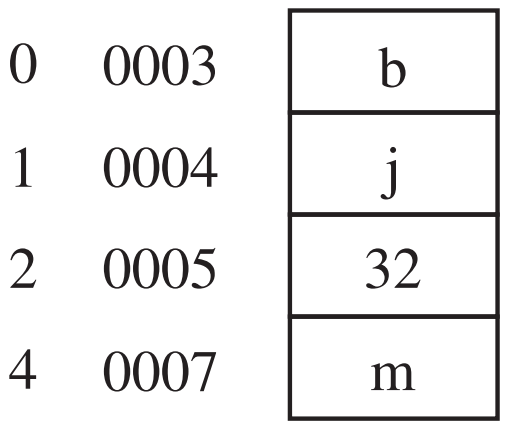
Figure 6.46 (continued)

Assembly Language

```
006C  496E69 msg0:    .ASCII  "Initials: \x00"
      746961
      6C733A
      2000
0077  416765 msg1:    .ASCII  "Age: \x00"
      3A2000
007D  47656E msg2:    .ASCII  "Gender: \x00"
      646572
      3A2000
0086  6D616C msg3:    .ASCII  "male\n\x00"
      650A00
008C  66656D msg4:    .ASCII  "female\n\x00"
      616C65
      0A00
0094                  .END
```



(a) A global structure at Level HOL6.



(b) The same global structure at Asmb5.

Linked data structure with a local pointer

- Equate the pointer field to its offset from the first byte of the node
- Allocate storage for the node with *tot* in the accumulator and a call to `malloc()`, where *tot* is the total number of bytes occupied by the structure

Linked data structure with a local pointer

- To get the field pointed to by `p`, generate `LDWX` with immediate addressing (`i`) to move the value of the field into the index register, followed by `LDWA` or `LDBA` from `p`, depending on the type in the cell, with stack-deferred indexed addressing (`sfx`).

High-Order Language

```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;
   struct node *next;
};

int main() {
   struct node *first, *p;
   int value;
   first = 0;
   scanf("%d", &value);
   while (value != -9999) {
      p = first;
      first = (struct node *) malloc(sizeof(struct node));
      first->data = value;
      first->next = p;
      scanf("%d", &value);
   }
   for (p = first; p != 0; p = p->next) {
      printf("%d ", p->data);
   }
   return 0;
}
```

Figure 6.48 (continued)

Assembly Language

```
0000  120003          BR      main
                data:   .EQUATE 0           ;struct field #2d
                next:   .EQUATE 2           ;struct field #2h
                ;
                ;******* main ()
                first:  .EQUATE 4           ;local variable #2h
                p:      .EQUATE 2           ;local variable #2h
                value:  .EQUATE 0           ;local variable #2d
0003  580006    main:   SUBSP   6,i         ;push #first #p #value
0006  C00000            LDWA    0,i         ;first = 0
0009  E30004            STWA    first,s
000C  330000            DECI    value,s     ;scanf("%d", &value);
000F  C30000    while:  LDWA    value,s     ;while (value != -9999)
0012  A0D8F1            CPWA    -9999,i
0015  18003F            BREQ    endWh
0018  C30004            LDWA    first,s     ;p = first
001B  E30002            STWA    p,s
001E  C00004            LDWA    4,i         ;first = (struct node *)
                                            ;  malloc(sizeof(struct node))
0021  24006A            CALL    malloc      ;allocate #data #next
0024  EB0004            STWX    first,s
0027  C30000            LDWA    value,s     ;first->data = value
002A  C80000            LDWX    data,i
002D  E70004            STWA    first,sfx
```

Figure 6.48 (continued)

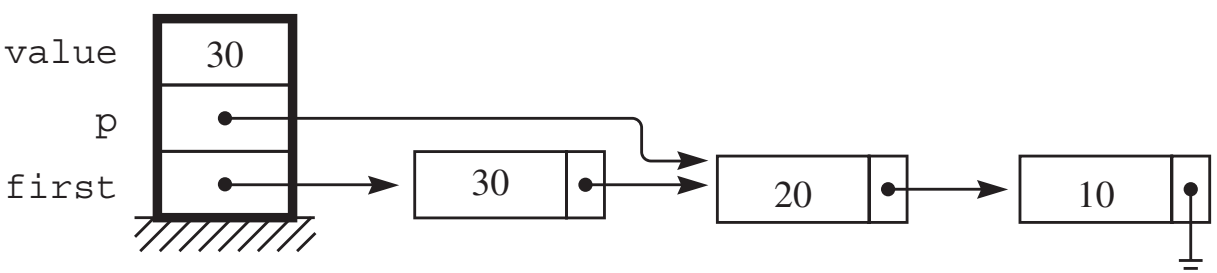
Assembly Language

```
0030  C30002          LDWA    p,s         ;first->next = p
0033  C80002          LDWX    next,i
0036  E70004          STWA    first,sfx
0039  330000          DECI    value,s     ;scanf("%d", &value)
003C  12000F          BR      while
003F  C30004 endWh:   LDWA    first,s     ;for (p = first
0042  E30002          STWA    p,s
0045  C30002 for:     LDWA    p,s         ;p != 0
0048  A00000          CPWA    0,i
004B  180066          BREQ    endFor
004E  C80000          LDWX    data,i      ;printf("%d ", p->data)
0051  3F0002          DECO    p,sfx
0054  D00020          LDBA    ' ',i
0057  F1FC16          STBA    charOut,d
005A  C80002          LDWX    next,i      ;p = p->next)
005D  C70002          LDWA    p,sfx
0060  E30002          STWA    p,s
0063  120045          BR      for
0066  500006 endFor:  ADDSP   6,i         ;pop #value #p #first
0069  00              STOP
```

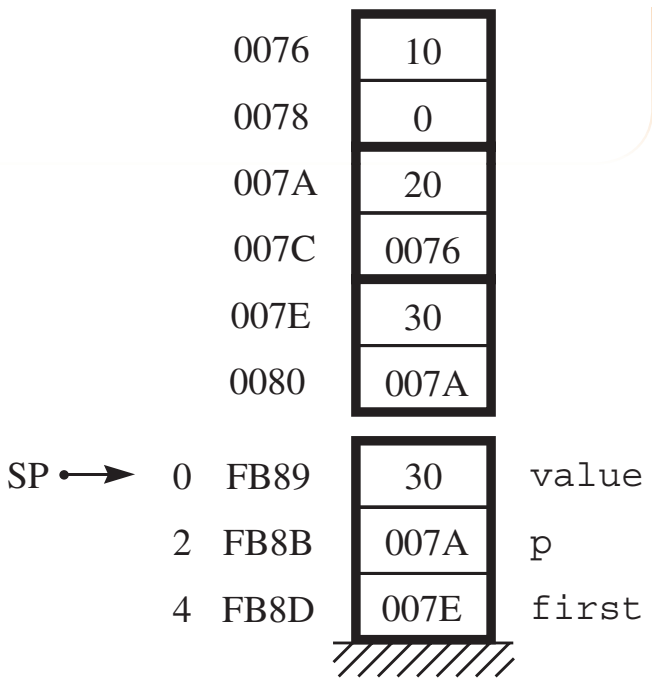
Assembly Language

```
                ;******* malloc()
                ;       Precondition: A contains number of bytes
                ;       Postcondition: X contains pointer to bytes
006A  C90074 malloc:  LDWX    hpPtr,d     ;returned pointer
006D  610074          ADDA    hpPtr,d     ;allocate from heap
0070  E10074          STWA    hpPtr,d     ;update hpPtr
0073  01              RET
0074  0076   hpPtr:   .ADDRSS heap        ;address of next free byte
0076  00      heap:   .BLOCK  1           ;first byte in the heap
0077                  .END
```

Figure 6.49



(a) The linked list at Level HOL6.



(b) The same linked list at Level Asmb5.

Linked data structure with a global pointer

- A problem for the student
- Exercise 6.10
- Problem 6.36

Problem 6.23

```
int binCoeff(int n, int k) {
   if ((k == 0) || (n == k)) {
      return 1;
   }
   else {
      return binCoeff(n - 1, k) + binCoeff(n - 1, k - 1);
   }
}
```