Compiling to the Assembly Level

Stack-relative addressing

- Oprnd = Mem[SP + OprndSpec]
- Asmb5 letter: `s`

The add SP instruction

- Instruction specifier: 0110 0aaa
- Mnemonic: `ADDSP`
- Adds the operand to the stack pointer, SP

$$\mathrm{SP} \leftarrow \mathrm{SP} + \mathrm{Oprnd}\ ;\ \mathrm{N} \leftarrow \mathrm{r} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{r} = 0\ ,$$

$$\mathrm{V} \leftarrow \{overflow\}\ ,\ \mathrm{C} \leftarrow \{carry\}$$

The subtract SP instruction

- Instruction specifier: 1000 1aaa
- Mnemonic: `SUBSP`
- Subtracts the operand from the stack pointer, SP

$$SP \leftarrow SP - Oprnd\ ;\ N \leftarrow r < 0\ ,\ Z \leftarrow r = 0\ ,$$

$$V \leftarrow \{overflow\}\ ,\ C \leftarrow \{carry\}$$

Pep/8 execute option

- SP←Mem[FFF8]
- PC←0000

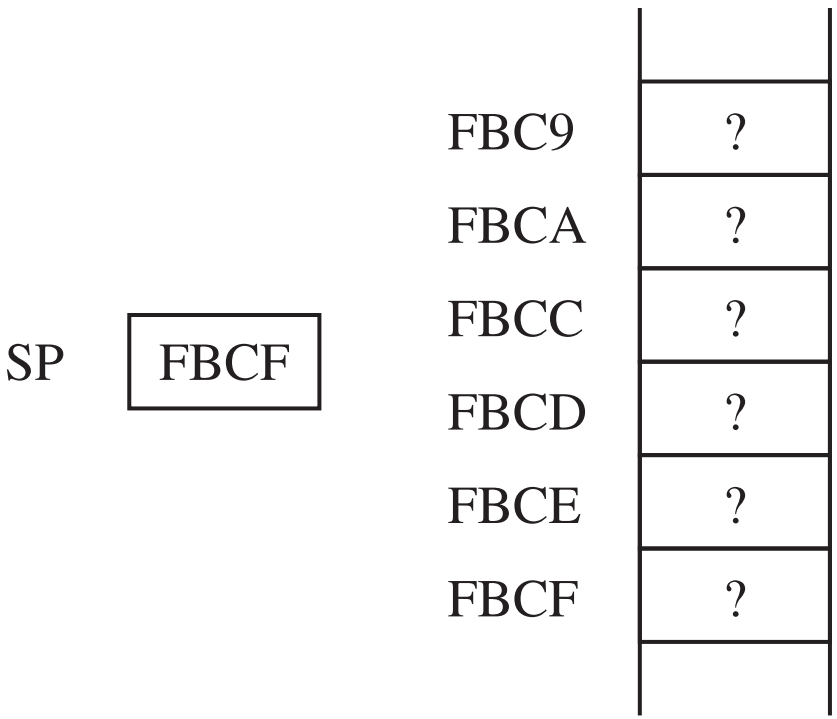
Figure 6.1

```
0000   C00042 LDA      'B',i          ;push B
0003   F3FFFF STBYTEA -1,s
0006   C0004D LDA      'M',i          ;push M
0009   F3FFFE STBYTEA -2,s
000C   C00057 LDA      'W',i          ;push W
000F   F3FFFD STBYTEA -3,s
0012   C0014F LDA      335,i          ;push 335
0015   E3FFFB STA      -5,s
0018   C00069 LDA      'i',i          ;push i
001B   F3FFFA STBYTEA -6,s
001E   680006 SUBSP    6,i            ;6 bytes on the run-time stack
0021   530005 CHARO    5,s            ;output B
0024   530004 CHARO    4,s            ;output M
0027   530003 CHARO    3,s            ;output W
002A   3B0001 DECO     1,s            ;output 335
002D   530000 CHARO    0,s            ;output i
0030   600006 ADDSP    6,i            ;deallocate stack storage
0033   00     STOP
0034          .END
```

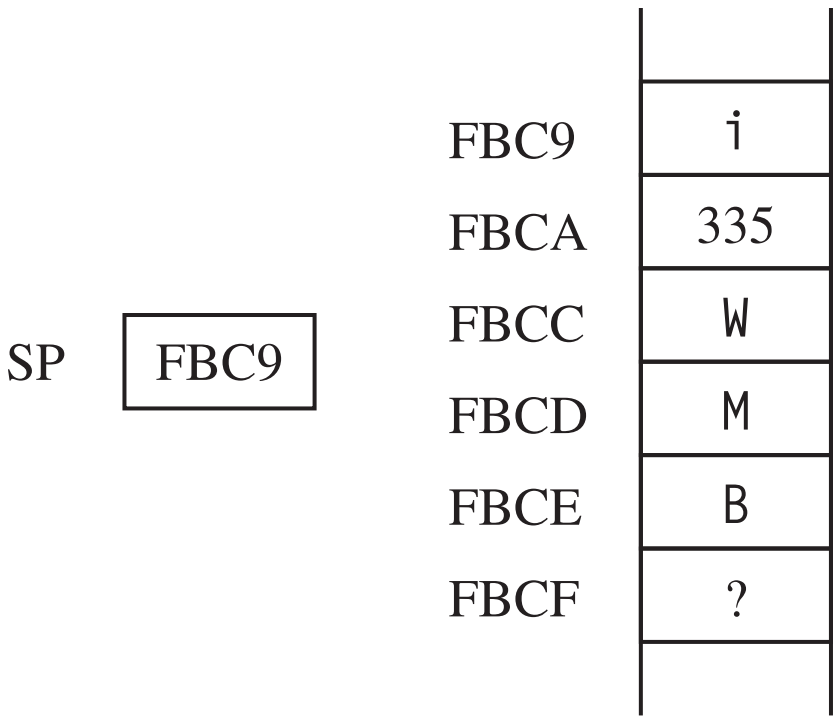
Output

```
BMW335i
```

Figure 6.2

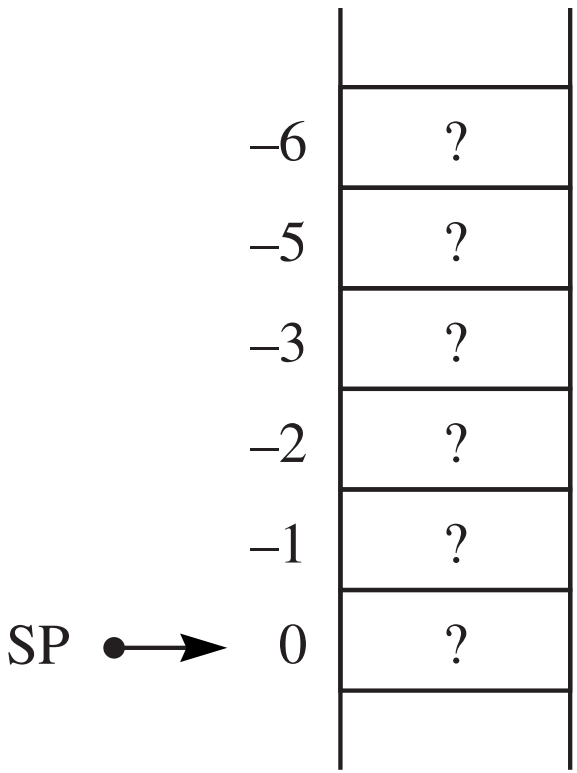


(a) Before the program executes.

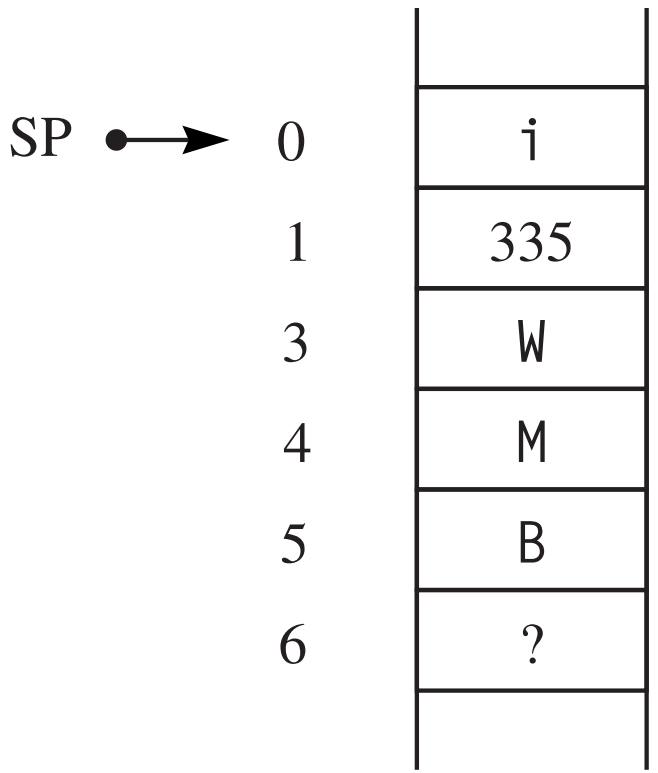


(b) After `SUBSP` executes.

Figure 6.3



(a) Before the program executes.



(b) After SUBSP executes.

Local variables

- Allocate locals with `SUBSP`
- Access locals with stack-relative addressing (`s`)
- Deallocate locals with `ADDSP`

Figure 6.4

High-Order Language

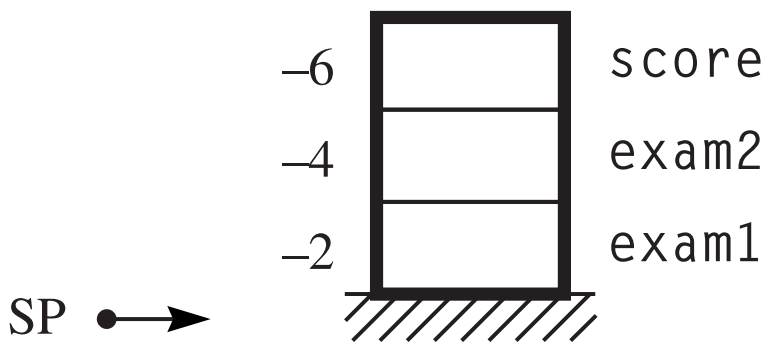
```
#include <iostream>
using namespace std;

int main () {
   const int bonus = 5;
   int exam1;
   int exam2;
   int score;
   cin >> exam1 >> exam2;
   score = (exam1 + exam2) / 2 + bonus;
   cout << "score = " << score << endl;
   return 0;
}
```

Figure 6.4
(Continued)

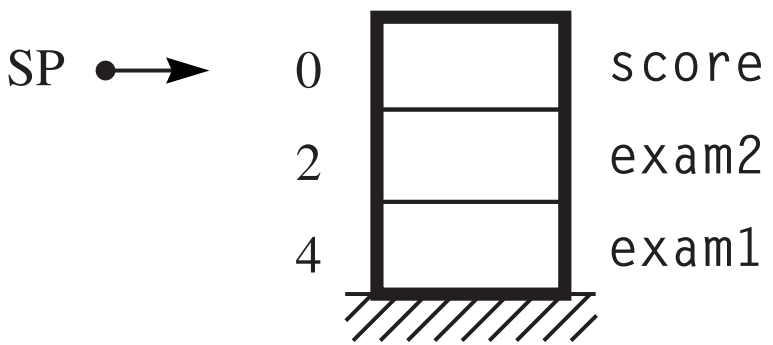
Assembly Language

```
0000  040003          BR      main
            bonus:    .EQUATE 5               ;constant
            exam1:    .EQUATE 4               ;local variable #2d
            exam2:    .EQUATE 2               ;local variable #2d
            score:    .EQUATE 0               ;local variable #2d
            ;
0003  680006 main:    SUBSP   6,i             ;allocate #exam1 #exam2 #score
0006  330004          DECI    exam1,s         ;cin >> exam1
0009  330002          DECI    exam2,s         ;    >> exam2
000C  C30004          LDA     exam1,s         ;score = (exam1
000F  730002          ADDA    exam2,s         ;    + exam2)
0012  1E              ASRA                    ;    / 2
0013  700005          ADDA    bonus,i         ;    + bonus
0016  E30000          STA     score,s
0019  410026          STRO    msg,d           ;cout << "score = "
001C  3B0000          DECO    score,s         ;    << score
001F  50000A          CHARO   '\n',i          ;    << endl
0022  600006          ADDSP   6,i             ;deallocate #score #exam2 #exam1
0025  00              STOP
0026  73636F msg:     .ASCII  "score = \x00"
      726520
      3D2000
002F                  .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` $C = 1 \Rightarrow PC \leftarrow Oprnd$

Figure 6.6

High-Order Language

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

int main () {
   int number;
   cin >> number;
   if (number < 0) {
      number = -number;
   }
   cout << number;
   return 0;
}
```

Figure 6.6
(Continued)

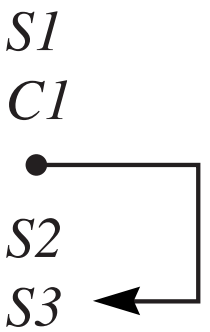
Assembly Language

```
0000   040003           BR      main
               number:  .EQUATE 0               ;local variable #2d
               ;
0003   680002 main:     SUBSP   2,i             ;allocate #number
0006   330000           DECI    number,s        ;cin >> number
0009   C30000 if:       LDA     number,s        ;if (number < 0)
000C   0E0016           BRGE    endIf
000F   C30000           LDA     number,s        ;    number = -number
0012   1A               NEGA
0013   E30000           STA     number,s
0016   3B0000 endIf:    DECO    number,s        ;cout << number
0019   600002           ADDSP   2,i             ;deallocate #number
001C   00               STOP
001D                    .END
```

Figure 6.7

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

(a) The structure at Level HOL6.



(b) The structure at level Asmb5 for Figure 6.6.

Optimizing compiler

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

The compare instruction

- Instruction specifier: 1011 raaa
- Mnemonic: `CPr` (`CPA`, `CPX`)
- Compare r with operand

$$\mathrm{T} \leftarrow \mathrm{r} - \mathrm{Oprnd}\ ;\ \mathrm{N} \leftarrow \mathrm{T} < 0\ ,\ \mathrm{Z} \leftarrow \mathrm{T} = 0\ ,$$

$$\mathrm{V} \leftarrow \{overflow\}\ ,\ \mathrm{C} \leftarrow \{carry\}$$

Figure 6.8

High-Order Language

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

int main () {
   const int limit = 100;
   int num;
   cin >> num;
   if (num >= limit) {
      cout << "high";
   }
   else {
      cout << "low";
   }
   return 0;
}
```

Figure 6.8
(Continued)

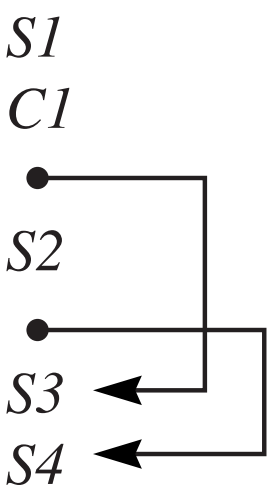
Assembly Language

```
0000  040003          BR      main
              limit:  .EQUATE 100             ;constant
              num:    .EQUATE 0               ;local variable #2d
              ;
0003  680002  main:   SUBSP   2,i             ;allocate #num
0006  330000          DECI    num,s           ;cin >> num
0009  C30000  if:     LDA     num,s           ;if (num >= limit)
000C  B00064          CPA     limit,i
000F  080018          BRLT    else
0012  41001F          STRO    msg1,d          ;   cout << "high"
0015  04001B          BR      endIf           ;else
0018  410024  else:   STRO    msg2,d          ;   cout << "low"
001B  600002  endIf:  ADDSP   2,i             ;deallocate #num
001E  00              STOP
001F  686967  msg1:   .ASCII  "high\x00"
      6800
0024  6C6F77  msg2:   .ASCII  "low\x00"
      00
0028                  .END
```

Figure 6.9

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

(a) The structure at Level HOL6.



(b) The structure at level Asmb5 for Figure 6.8.

Figure 6.10

High-Order Language

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

char letter;

int main () {
   cin >> letter;
   while (letter != '*') {
      cout << letter;
      cin >> letter;
   }
   return 0;
}
```

Figure 6.10
(Continued)

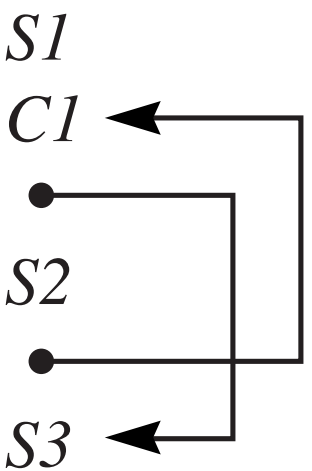
Assembly Language

```
0000   040004           BR       main
0003   00       letter: .BLOCK   1              ;global variable #1c
                ;
0004   490003   main:   CHARI    letter,d       ;cin >> letter
0007   C00000           LDA      0x0000,i
000A   D10003   while:  LDBYTEA  letter,d       ;while (letter != '*')
000D   B0002A           CPA      '*',i
0010   0A001C           BREQ     endWh
0013   510003           CHARO    letter,d       ;   cout << letter
0016   490003           CHARI    letter,d       ;   cin >> letter
0019   04000A           BR       while
001C   00       endWh:  STOP
001D                    .END
```

Figure 6.11

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

(a) The structure at level HOL6.



(b) The structure at level Asmb5 for Figure 6.10.

Figure 6.12

High-Order Language

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

int cop;
int driver;

int main () {
   cop = 0;
   driver = 40;
   do {
      cop += 25;
      driver += 20;
   }
   while (cop < driver);
   cout << cop;
   return 0;
}
```

Figure 6.12
(Continued)

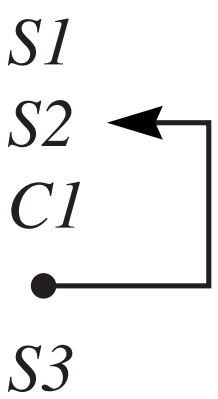
Assembly Language

```
0000  040007          BR      main
0003  0000    cop:    .BLOCK  2           ;global variable #2d
0005  0000    driver: .BLOCK  2           ;global variable #2d
              ;
0007  C00000  main:   LDA     0,i         ;cop = 0
000A  E10003          STA     cop,d
000D  C00028          LDA     40,i        ;driver = 40
0010  E10005          STA     driver,d
0013  C10003  do:     LDA     cop,d       ;   cop += 25
0016  700019          ADDA    25,i
0019  E10003          STA     cop,d
001C  C10005          LDA     driver,d    ;   driver += 20
001F  700014          ADDA    20,i
0022  E10005          STA     driver,d
0025  C10003  while:  LDA     cop,d       ;while (cop < driver)
0028  B10005          CPA     driver,d
002B  080013          BRLT    do
002E  390003          DECO    cop,d       ;cout << cop
0031  00              STOP
0032                  .END
```

Figure 6.13

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

(a) The structure at level HOL6.



(b) The structure at level Asmb5 for Figure 6.12.

The `for` loop

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

Figure 6.14

High-Order Language

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

int main () {
   int j;
   for (j = 0; j < 3; j++) {
      cout << "j = " << j << endl;
   }
   cout << "j = " << j << endl;
   return 0;
}
```

Figure 6.14
(Continued)

Assembly Language

```
0000  040003          BR      main
            j:        .EQUATE 0               ;local variable #2d
            ;
0003  680002 main:    SUBSP   2,i             ;allocate #j
0006  C00000          LDA     0,i             ;for (j = 0
0009  E30000          STA     j,s
000C  B00003 for:     CPA     3,i             ;   j < 3
000F  0E0027          BRGE    endFor
0012  410034          STRO    msg,d           ;   cout << "j = "
0015  3B0000          DECO    j,s             ;      << j
0018  50000A          CHARO   '\n',i          ;      << endl
001B  C30000          LDA     j,s             ;   j++)
001E  700001          ADDA    1,i
0021  E30000          STA     j,s
0024  04000C          BR      for
0027  410034 endFor:  STRO    msg,d           ;cout << "j = "
002A  3B0000          DECO    j,s             ;   << j
002D  50000A          CHARO   '\n',i          ;   << endl
0030  600002          ADDSP   2,i             ;deallocate #j
0033  00              STOP
0034  6A203D msg:     .ASCII  "j = \x00"
      2000
0039                  .END
```

Figure 6.15

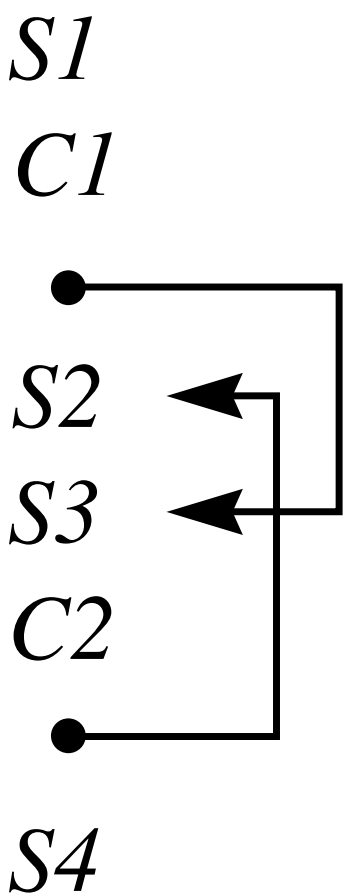


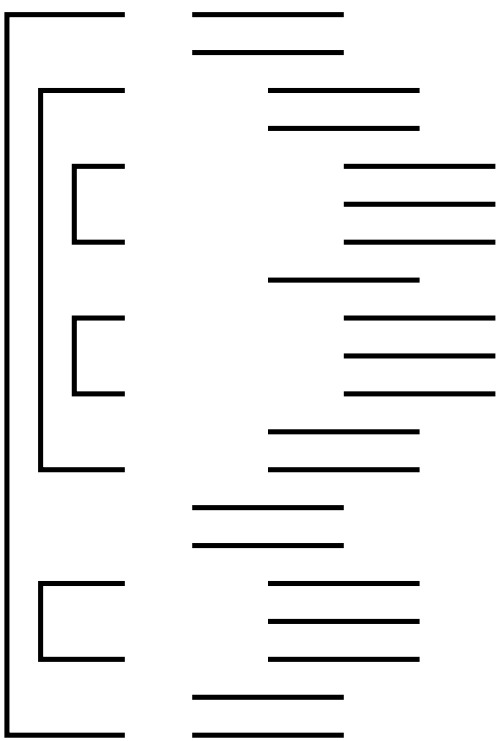
Figure 6.16

```
0000  040009          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          LDA     n2,d
0012  B10007          CPA     n3,d
0015  08002A          BRLT    L1
0018  310003          DECI    n1,d
001B  C10003          LDA     n1,d
001E  B10007          CPA     n3,d
0021  080074          BRLT    L7
0024  040065          BR      L6
0027  E10007          STA     n3,d
002A  310003  L1:     DECI    n1,d
002D  C10005          LDA     n2,d
0030  B10003          CPA     n1,d
0033  080053          BRLT    L5
0036  390003          DECO    n1,d
0039  390005          DECO    n2,d
003C  390007  L2:     DECO    n3,d
003F  00              STOP
```

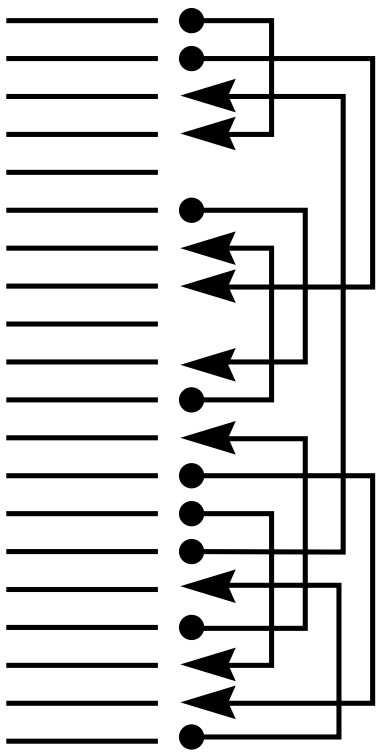
Figure 6.16
(Continued)

```
0040  390005 L3:     DECO    n2,d
0043  390007         DECO    n3,d
0046  040081         BR      L9
0049  390003 L4:     DECO    n1,d
004C  390005         DECO    n2,d
004F  00             STOP
0050  E10003         STA     n1,d
0053  C10007 L5:     LDA     n3,d
0056  B10003         CPA     n1,d
0059  080040         BRLT    L3
005C  390005         DECO    n2,d
005F  390003         DECO    n1,d
0062  04003C         BR      L2
0065  390007 L6:     DECO    n3,d
0068  C10003         LDA     n1,d
006B  B10005         CPA     n2,d
006E  080049         BRLT    L4
0071  04007E         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
```

Figure 6.17



(a) Structured flow.



(b) Spaghetti code.

The Structured Programming Theorem

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

Bohm and Jacopini, 1966

The goto controversy

... the quality of programmers is a decreasing function of the density of goto statements in the programs they produce. 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.

Dijkstra, 1968

The call subroutine instruction

- Instruction specifier: 0001 011a
- 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: 0101 1nnn
- Mnemonic: `RETn` (`RET0`, `RET1`, ..., `RET7`)
- Pop storage for locals and return address off of run-time stack

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

Figure 6.18

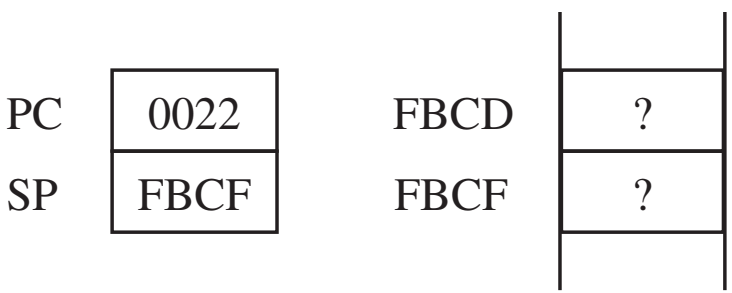
High-Order Language

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

void printTri () {
   cout << "*" << endl;
   cout << "**" << endl;
   cout << "***" << endl;
}

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

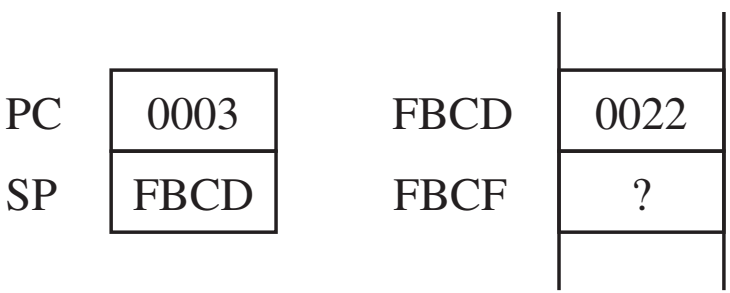
Figure 6.18
(Continued)

Assembly Language

```
0000  04001F          BR      main
                ;
                ;******* void printTri ()
0003  410016  printTri:STRO    msg1,d          ;cout << "*"
0006  50000A          CHARO   '\n',i          ;   << endl
0009  410018          STRO    msg2,d          ;cout << "**"
000C  50000A          CHARO   '\n',i          ;   << endl
000F  41001B          STRO    msg3,d          ;cout << "**"
0012  50000A          CHARO   '\n',i          ;   << endl
0015  58              RETO
0016  2A00    msg1:  .ASCII  "*\x00"
0018  2A2A00  msg2:  .ASCII  "**\x00"
001B  2A2A2A  msg3:  .ASCII  "***\x00"
      00
                ;
                ;******* int main ()
001F  160003  main:   CALL    printTri        ;printTri ()
0022  160003          CALL    printTri        ;printTri ()
0025  160003          CALL    printTri        ;printTri ()
0028  00              STOP
0029                  .END
```

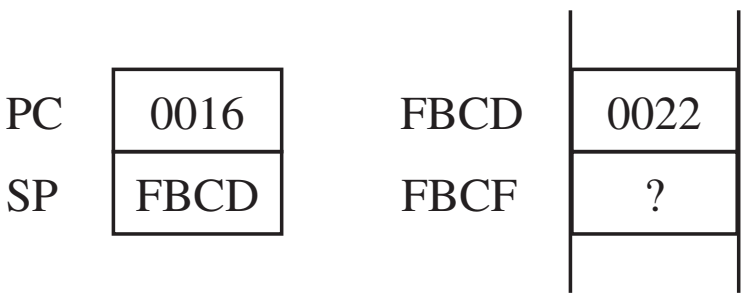


(a) Before execution of the first `CALL`.

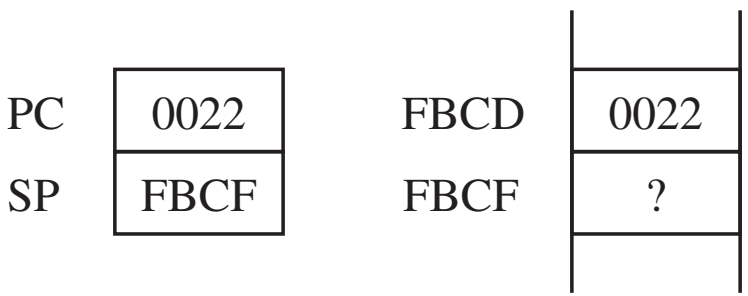


(b) After execution of the first `CALL`.

Figure 6.20



(a) Before the first execution of `RET0`.



(b) After the first execution of `RET0`.

To call a `void` function in C++

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

In assembly language

- Calling pushes actual parameters (executes `SUBSP`)
- Calling pushes return address (executes `CALL`)
- Called allocates local variables (executes `SUBSP`)
- Called executes its body.
- Called deallocates local variables and pops return address (executes `RETn`)
- Calling pops actual parameters (executes `ADDSP`)

Call-by-value with global variables

- To push the actual parameter, use `LDA` with direct addressing
- To access the formal parameter, use stack-relative addressing (`s`)

Figure 6.21

High-Order Language

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

int numPts;
int value;
int j;

void printBar (int n) {
   int k;
   for (k = 1; k <= n; k++) {
      cout << '*';
   }
   cout << endl;
}

int main () {
   cin >> numPts;
   for (j = 1; j <= numPts; j++) {
      cin >> value;
      printBar (value);
   }
   return 0;
}
```

Figure 6.21
(Continued)

Assembly Language

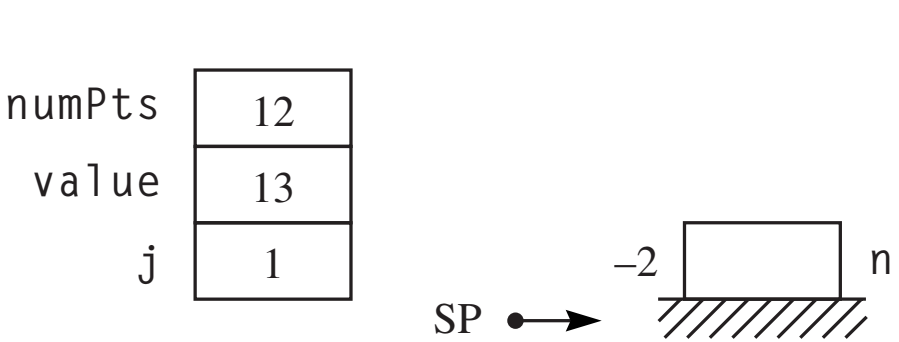
```
0000  04002B           BR      main
0003  0000   numPts:   .BLOCK  2               ;global variable #2d
0005  0000   value:    .BLOCK  2               ;global variable #2d
0007  0000   j:        .BLOCK  2               ;global variable #2d
             ;
             ;******* void printBar (int n)
             n:        .EQUATE 4               ;formal parameter #2d
             k:        .EQUATE 0               ;local variable #2d
0009  680002 printBar:SUBSP   2,i             ;allocate #k
000C  C00001           LDA     1,i             ;for (k = 1
000F  E30000           STA     k,s
0012  B30004 for1:     CPA     n,s             ;k <= n
0015  100027           BRGT    endFor1
0018  50002A           CHARO   '*',i           ;   cout << '*'
001B  C30000           LDA     k,s             ;k++)
001E  700001           ADDA    1,i
0021  E30000           STA     k,s
0024  040012           BR      for1
0027  50000A endFor1:  CHARO   '\n',i          ;cout << endl
002A  5A               RET2                    ;deallocate #k, pop retAddr
```

Figure 6.21
(Continued)

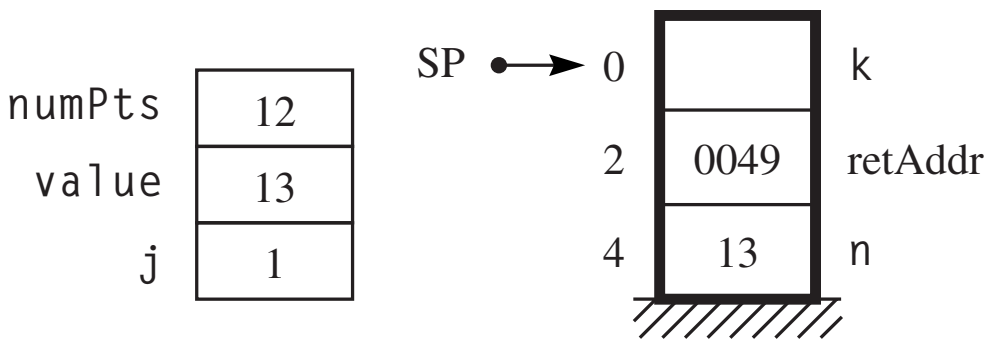
Assembly Language

```
                ;******* main ()
002B   310003 main:    DECI    numPts,d      ;cin >> numPts
002E   C00001          LDA     1,i           ;for (j = 1
0031   E10007          STA     j,d
0034   B10003 for2:    CPA     numPts,d      ;j <= numPts
0037   100058          BRGT    endFor2
003A   310005          DECI    value,d       ;   cin >> value
003D   C10005          LDA     value,d       ;   call by value
0040   E3FFFE          STA     -2,s
0043   680002          SUBSP   2,i           ;   push #n
0046   160009          CALL    printBar      ;   push retAddr
0049   600002          ADDSP   2,i           ;   pop #n
004C   C10007          LDA     j,d           ;j++)
004F   700001          ADDA    1,i
0052   E10007          STA     j,d
0055   040034          BR      for2
0058   00      endFor2: STOP
0059                   .END
```

Figure 6.22



(a) After `cin >> value`.



(b) After allocation with `SUBSP` in `printBar`.

Call-by-value with local variables

- To push the actual parameter, use `LDA` with stack-relative addressing
- To access the formal parameter, use stack-relative addressing (`s`)

Figure 6.23

High-Order Language

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

void printBar (int n) {
   int k;
   for (k = 1; k <= n; k++) {
      cout << '*';
   }
   cout << endl;
}

int main () {
   int numPts;
   int value;
   int j;
   cin >> numPts;
   for (j = 1; j <= numPts; j++) {
      cin >> value;
      printBar (value);
   }
   return 0;
}
```

Figure 6.23
(Continued)

Assembly Language

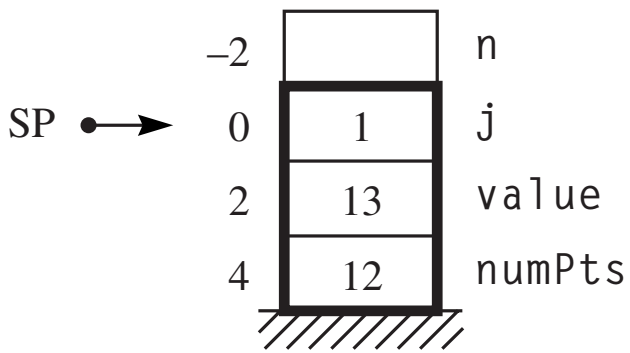
```
0000  040025          BR      main
                ;
                ;******* void printBar (int n)
                n:      .EQUATE 4               ;formal parameter #2d
                k:      .EQUATE 0               ;local variable #2d
0003  680002 printBar:SUBSP   2,i             ;allocate #k
0006  C00001          LDA     1,i             ;for (k = 1
0009  E30000          STA     k,s
000C  B30004 for1:    CPA     n,s             ;k <= n
000F  100021          BRGT    endFor1
0012  50002A          CHARO   '*',i           ;   cout << '*'
0015  C30000          LDA     k,s             ;k++)
0018  700001          ADDA    1,i
001B  E30000          STA     k,s
001E  04000C          BR      for1
0021  50000A endFor1: CHARO   '\n',i          ;cout << endl
0024  5A              RET2                    ;deallocate #k, pop retAddr
```

Figure 6.23
(Continued)

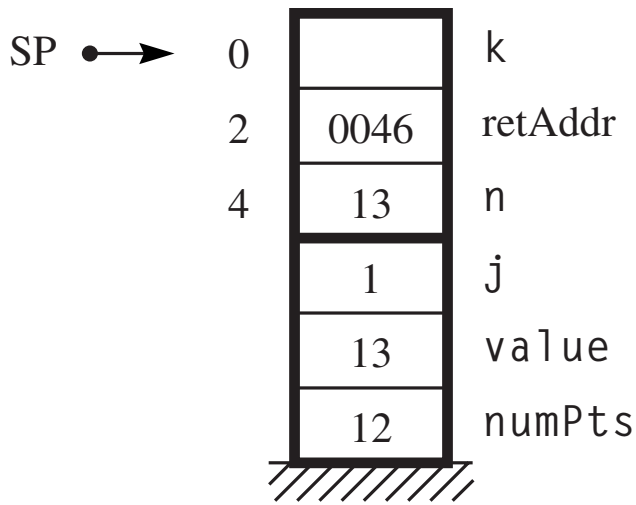
Assembly Language

```
                ;******* main ()
                numPts:   .EQUATE 4               ;local variable #2d
                value:    .EQUATE 2               ;local variable #2d
                j:        .EQUATE 0               ;local variable #2d
0025  680006    main:     SUBSP   6,i             ;allocate #numPts #value #j
0028  330004              DECI    numPts,s        ;cin >> numPts
002B  C00001              LDA     1,i             ;for (j = 1
002E  E30000              STA     j,s
0031  B30004    for2:     CPA     numPts,s        ;j <= numPts
0034  100055              BRGT    endFor2
0037  330002              DECI    value,s         ;   cin >> value
003A  C30002              LDA     value,s         ;   call by value
003D  E3FFFE              STA     -2,s
0040  680002              SUBSP   2,i             ;   push #n
0043  160003              CALL    printBar        ;   push retAddr
0046  600002              ADDSP   2,i             ;   pop #n
0049  C30000              LDA     j,s             ;j++)
004C  700001              ADDA    1,i
004F  E30000              STA     j,s
0052  040031              BR      for2
0055  600006    endFor2:  ADDSP   6,i             ;deallocate #j #value #numPts
0058  00                  STOP
0059                      .END
```

Figure 6.24



(a) After `cin >> value`.



(b) After allocation with `SUBSP` in `printBar`.

To call a non-`void` function in C++

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

Figure 6.25

High-Order Language

```
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 () {
   cout << "binCoeff (3, 1) = " << binCoeff (3, 1); // ra1
   cout << endl;
   return 0;
}
```

Figure 6.25
(Continued)

Assembly Language

```
0000  040065          BR      main
                ;
                ;******* int binomCoeff (int n, int k)
                retVal: .EQUATE 10              ;returned 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  680004 binCoeff:SUBSP   4,i               ;allocate #y1 #y2
0006  C30006 if:      LDA     k,s               ;if ((k == 0)
0009  0A0015          BREQ    then
000C  C30008          LDA     n,s               ;|| (n == k))
000F  B30006          CPA     k,s
0012  0C001C          BRNE    else
0015  C00001 then:    LDA     1,i               ;return 1
0018  E3000A          STA     retVal,s
001B  5C              RET4                      ;deallocate #y2 #y1, pop retAddr
```

Figure 6.25
(Continued)

Assembly Language

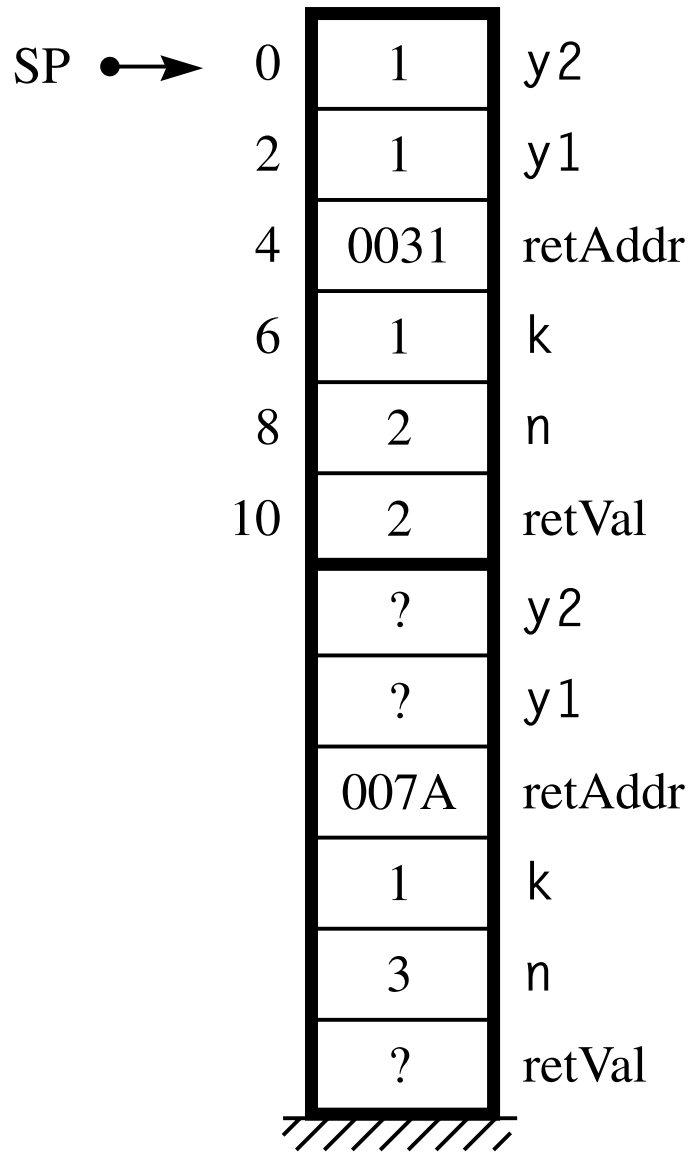
```
001C  C30008 else:   LDA     n,s          ;push n - 1
001F  800001         SUBA    1,i
0022  E3FFFC         STA     -4,s
0025  C30006         LDA     k,s          ;push k
0028  E3FFFA         STA     -6,s
002B  680006         SUBSP   6,i          ;push #retVal #n #k
002E  160003         CALL    binCoeff     ;binomCoeff (n - 1, k)
0031  600006 ra2:    ADDSP   6,i          ;pop #k #n #retVal
0034  C3FFFE         LDA     -2,s         ;y1 = binomCoeff (n - 1, k)
0037  E30002         STA     y1,s
003A  C30008         LDA     n,s          ;push n - 1
003D  800001         SUBA    1,i
0040  E3FFFC         STA     -4,s
0043  C30006         LDA     k,s          ;push k - 1
0046  800001         SUBA    1,i
0049  E3FFFA         STA     -6,s
004C  680006         SUBSP   6,i          ;push #retVal #n #k
004F  160003         CALL    binCoeff     ;binomCoeff (n - 1, k - 1)
0052  600006 ra3:    ADDSP   6,i          ;pop #k #n #retVal
0055  C3FFFE         LDA     -2,s         ;y2 = binomCoeff (n - 1, k - 1)
0058  E30000         STA     y2,s
005B  C30002         LDA     y1,s         ;return y1 + y2
005E  730000         ADDA    y2,s
0061  E3000A         STA     retVal,s
0064  5C     endIf:  RET4                 ;deallocate #y2 #y1, pop retAddr
```

Figure 6.25
(Continued)

Assembly Language

```
                ;******* main ()
0065   410084 main:    STRO    msg,d           ;cout << "binCoeff (3, 1) = "
0068   C00003          LDA     3,i             ;push 3
006B   E3FFFC          STA     -4,s
006E   C00001          LDA     1,i             ;push 1
0071   E3FFFA          STA     -6,s
0074   680006          SUBSP   6,i             ;push #retVal #n #k
0077   160003          CALL    binCoeff        ;binomCoeff (3, 1)
007A   600006 ra1:     ADDSP   6,i             ;pop #k #n #retVal
007D   3BFFFE          DECO    -2,s            ;<< binCoeff (3, 1)
0080   50000A          CHARO   '\n',i          ;cout << endl
0083   00              STOP
0084   62696E msg:     .ASCII  "binCoeff (3, 1) = \x00"
       436F65
       666620
       28332C
       203129
       203D20
       00
0097                   .END
```

Figure 6.26



Stack-relative deferred addressing

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

Call-by-reference with global variables

- To push the actual parameter, use `LDA` with immediate addressing
- To access the formal parameter, use stack-relative deferred addressing (`sf`)

Figure 6.27

High-Order Language

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

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 () {
   cout << "Enter an integer: ";
   cin >> a;
   cout << "Enter an integer: ";
   cin >> b;
   order (a, b);
   cout << "Ordered they are: " << a << ", " << b << endl; // ra1
   return 0;
}
```

Figure 6.27
(Continued)

Assembly Language

```
0000  04003C          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  680002  swap:   SUBSP   2,i             ;allocate #temp
000A  C40006          LDA     r,sf            ;temp = r
000D  E30000          STA     temp,s
0010  C40004          LDA     s,sf            ;r = s
0013  E40006          STA     r,sf
0016  C30000          LDA     temp,s          ;s = temp
0019  E40004          STA     s,sf
001C  5A              RET2                    ;deallocate #temp, pop retAddr
```

Figure 6.27
(Continued)

Assembly Language

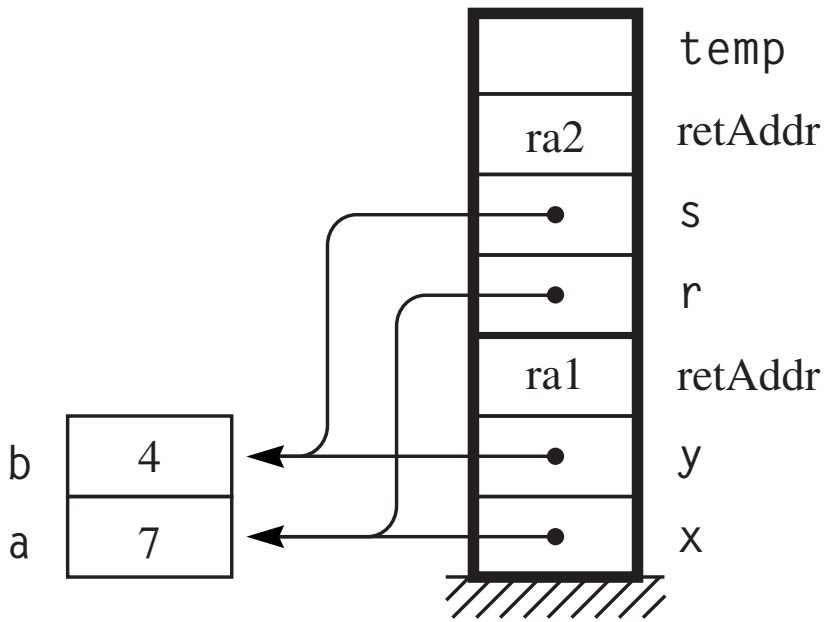
```
                ;******* void order (int& x, int& y)
                x:      .EQUATE 4               ;formal parameter #2h
                y:      .EQUATE 2               ;formal parameter #2h
001D  C40004    order:  LDA     x,sf            ;if (x > y)
0020  B40002            CPA     y,sf
0023  06003B            BRLE    endIf
0026  C30004            LDA     x,s             ;   push x
0029  E3FFFE            STA     -2,s
002C  C30002            LDA     y,s             ;   push y
002F  E3FFFC            STA     -4,s
0032  680004            SUBSP   4,i             ;   push #r #s
0035  160007            CALL    swap            ;   swap (x, y)
0038  600004            ADDSP   4,i             ;   pop #s #r
003B  58        endIf:  RET0                    ;pop retAddr
```

Figure 6.27
(Continued)

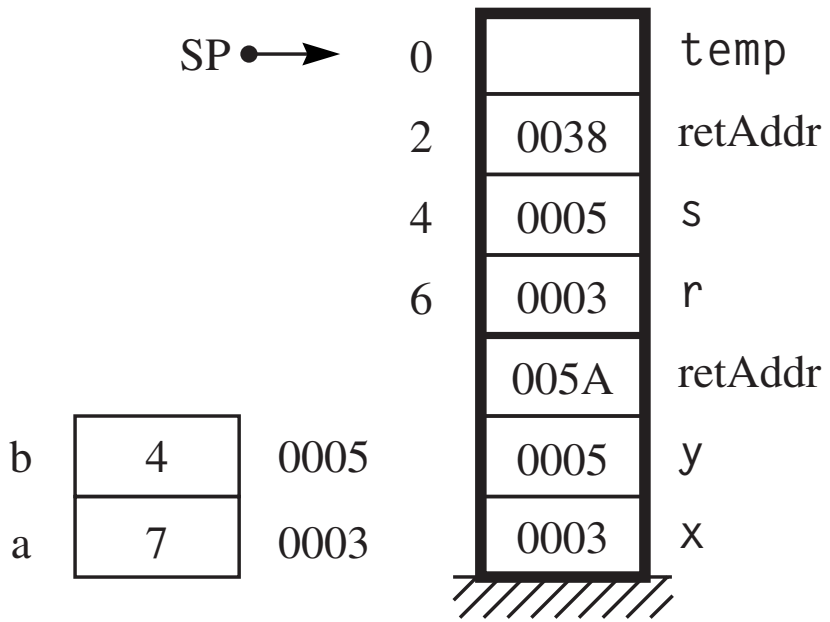
Assembly Language

```
                ;******* main ()
003C  41006D main:    STRO    msg1,d          ;cout << "Enter an integer: "
003F  310003          DECI    a,d             ;cin >> a
0042  41006D          STRO    msg1,d          ;cout << "Enter an integer: "
0045  310005          DECI    b,d             ;cin >> b
0048  C00003          LDA     a,i             ;push the address of a
004B  E3FFFE          STA     -2,s
004E  C00005          LDA     b,i             ;push the address of b
0051  E3FFFC          STA     -4,s
0054  680004          SUBSP   4,i             ;push #x #y
0057  16001D          CALL    order           ;order (a, b)
005A  600004 ra1:     ADDSP   4,i             ;pop #y #x
005D  410080          STRO    msg2,d          ;cout << "Ordered they are: "
0060  390003          DECO    a,d             ;     << a
0063  410093          STRO    msg3,d          ;     << ", "
0066  390005          DECO    b,d             ;     << b
0069  50000A          CHARO   '\n',i          ;     << endl
006C  00              STOP
006D  456E74 msg1:    .ASCII  "Enter an integer: \x00"
      ...
0080  4F7264 msg2:    .ASCII  "Ordered they are: \x00"
      ...
0093  2C2000 msg3:    .ASCII  ", \x00"
0096                  .END
```

Figure 6.28



(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 0010
- Mnemonic: `MOVSPA`
- Accumulator gets the stack pointer

$$A \leftarrow SP$$

Call-by-reference with local variables

- To push the actual parameter, use unary `MOVSPA` followed by `ADDA` with immediate addressing
- To access the formal parameter, use stack-relative deferred addressing (`sf`)

Figure 6.29

High-Order Language

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

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

int main () {
   int perim, width, height;
   cout << "Enter width: ";
   cin >> width;
   cout << "Enter height: ";
   cin >> height;
   rect (perim, width, height);
   // ra1
   cout << "perim = " << perim << endl;
   return 0;
}
```

Figure 6.29
(Continued)

Assembly Language

```
0000  04000E          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:   LDA     w,s               ;p = (w + h) * 2
0006  730002          ADDA    h,s
0009  1C              ASLA
000A  E40006          STA     p,sf
000D  58              RET0                      ;pop retAddr
```

Figure 6.29
(Continued)

Assembly Language

```
                ;******* main ()
                perim:   .EQUATE 4           ;local variable #2d
                width:   .EQUATE 2           ;local variable #2d
                height:  .EQUATE 0           ;local variable #2d
000E  680006    main:    SUBSP   6,i         ;allocate #perim #width #height
0011  410046             STRO    msg1,d      ;cout << "Enter width: "
0014  330002             DECI    width,s     ;cin >> width
0017  410054             STRO    msg2,d      ;cout << "Enter height: "
001A  330000             DECI    height,s    ;cin >> height
001D  02                 MOVSPA              ;push the address of perim
001E  700004             ADDA    perim,i
0021  E3FFFE             STA     -2,s
0024  C30002             LDA     width,s     ;push the value of width
0027  E3FFFC             STA     -4,s
002A  C30000             LDA     height,s    ;push the value of height
002D  E3FFFA             STA     -6,s
0030  680006             SUBSP   6,i         ;push #p #w #h
0033  160003             CALL    rect        ;rect (perim, width, height)
0036  600006    ra1:     ADDSP   6,i         ;pop #h #w #p
0039  410063             STRO    msg3,d      ;cout << "perim = "
003C  3B0004             DECO    perim,s     ;     << perim
003F  50000A             CHARO   '\n',i      ;     << endl
0042  600006             ADDSP   6,i         ;deallocate #height #width #perim
0045  00                 STOP
```

Figure 6.29
(Continued)

Assembly Language

```
0046   456E74 msg1:    .ASCII   "Enter width: \x00"
       ...
0054   456E74 msg2:    .ASCII   "Enter height: \x00"
       ...
0063   706572 msg3:    .ASCII   "perim = \x00"
       ...
006C                   .END
```

Figure 6.30
Stack at Level HOL6

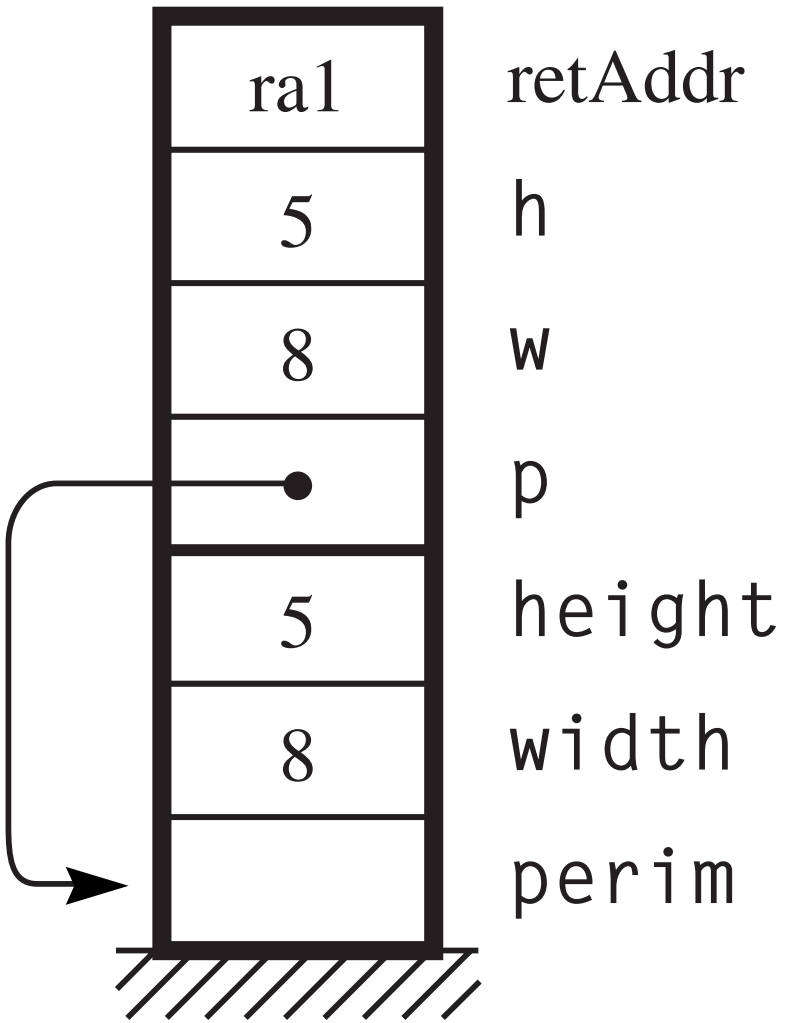
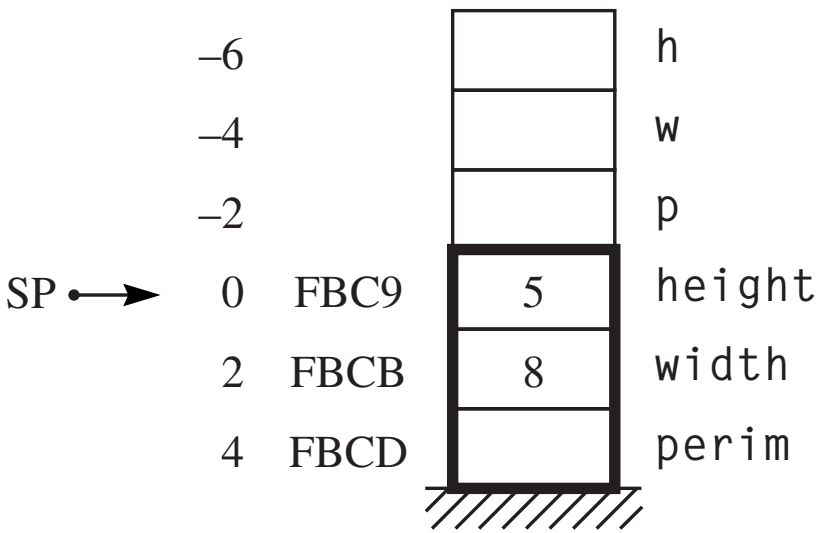
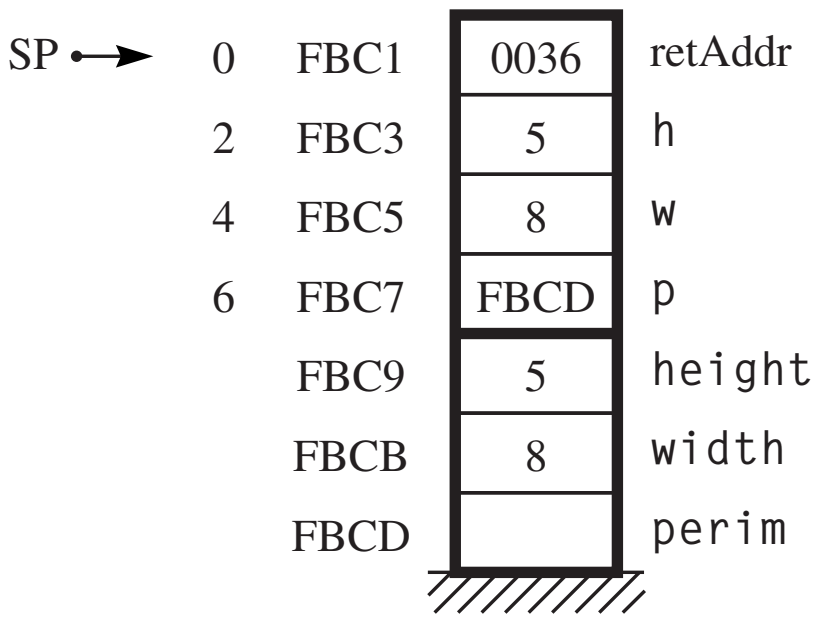


Figure 6.31
Stack at level ASMB5



(a) Before the procedure call.



(b) After the procedure call.

Boolean types

```
true:   .EQUATE 1
false:  .EQUATE 0
```

Figure 6.32

High-Order Language

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

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;
   cin >> age;
   if (inRange (age)) {
      cout << "Qualified\n";
   }
   else {
      cout << "Unqualified\n";
   }
   return 0;
}
```

Figure 6.32
(Continued)

Assembly Language

```
0000  040023           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: LDA     LOWER,i       ;if ((LOWER <= a)
0006  B30002 if:      CPA     a,s
0009  10001C          BRGT    else
000C  C30002          LDA     a,s           ;   && (a <= UPPER))
000F  B00041          CPA     UPPER,i
0012  10001C          BRGT    else
0015  C00001 then:    LDA     true,i        ;   return true
0018  E30004          STA     retVal,s
001B  58              RETO
001C  C00000 else:    LDA     false,i       ;   return false
001F  E30004          STA     retVal,s
0022  58              RETO
```

Figure 6.32
(Continued)

Assembly Language

```
                ;******* main ()
                age:    .EQUATE 0           ;local variable #2d
0023  680002    main:   SUBSP   2,i          ;allocate #age
0026  330000            DECI    age,s       ;cin >> age
0029  C30000    if2:    LDA     age,s       ;if (
002C  E3FFFC            STA     -4,s        ;store the value of age
002F  680004            SUBSP   4,i         ;push #retVal #a
0032  160003            CALL    inRange     ;   (inRange (age))
0035  600004            ADDSP   4,i         ;pop #a #retVal
0038  C3FFFE            LDA     -2,s        ;load retVal
003B  0A0044            BREQ    else2       ;branch if retVal == false (i.e. 0)
003E  41004B    then2:  STRO    msg1,d      ;   cout << "Qualified\n"
0041  040047            BR      endif2
0044  410056    else2:  STRO    msg2,d      ;   cout << "Unqualified\n"
0047  600002    endif2: ADDSP   2,i         ;deallocate #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-indexed deferred	111	sxf	Mem [Mem [SP + OprndSpec] + X]

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

Indexed addressing

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

Global arrays

- Allocate total number of bytes of `v` with `.BLOCK`
- To access element `v[i]`, load `i` into index register, multiply by number of bytes per cell, use indexed addressing (`x`)

Figure 6.34

High-Order Language

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

int vector[4];
int j;

int main () {
   for (j = 0; j < 4; j++) {
      cin >> vector[j];
   }
   for (j = 3; j >= 0; j--) {
      cout << j  << ' ' << vector[j] << endl;
   }
   return 0;
}
```

Figure 6.34
(Continued)

Assembly Language

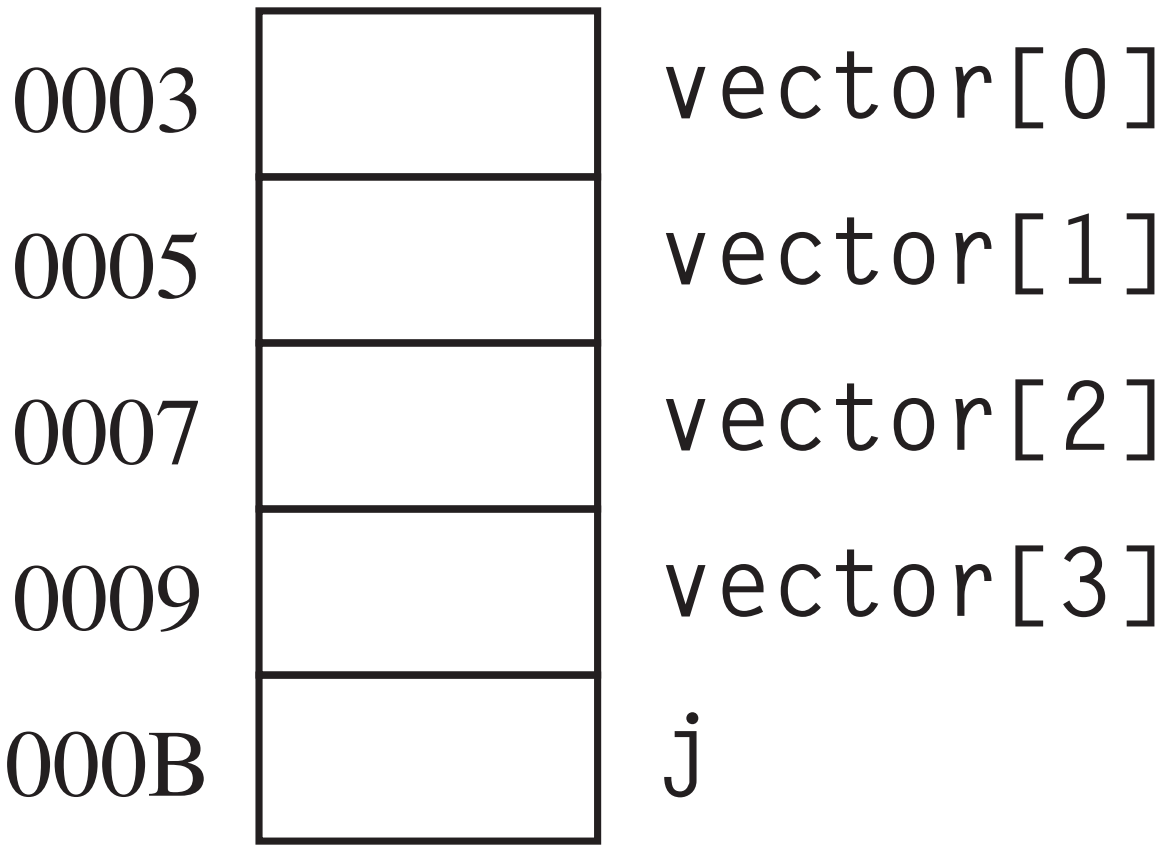
```
0000   04000D            BR      main
0003   000000 vector:    .BLOCK  8               ;global variable #2d4a
       000000
       0000
000B   0000   j:         .BLOCK  2               ;global variable #2d
```

Figure 6.34
(Continued)

Assembly Language

```
                ;******* main ()
000D  C80000 main:    LDX     0,i             ;for (j = 0
0010  E9000B          STX     j,d
0013  B80004 for1:    CPX     4,i             ;   j < 4
0016  0E0029          BRGE    endFor1
0019  1D              ASLX                    ;   an integer is two bytes
001A  350003          DECI    vector,x        ;   cin >> vector[j]
001D  C9000B          LDX     j,d             ;   j++)
0020  780001          ADDX    1,i
0023  E9000B          STX     j,d
0026  040013          BR      for1
0029  C80003 endFor1: LDX     3,i             ;for (j = 3
002C  E9000B          STX     j,d
002F  B80000 for2:    CPX     0,i             ;   j >= 0
0032  08004E          BRLT    endFor2
0035  39000B          DECO    j,d             ;   cout << j
0038  500020          CHARO   ' ',i           ;      << ' '
003B  1D              ASLX                    ;   an integer is two bytes
003C  3D0003          DECO    vector,x        ;      << vector[j]
003F  50000A          CHARO   '\n',i          ;      << endl
0042  C9000B          LDX     j,d             ;   j--)
0045  880001          SUBX    1,i
0048  E9000B          STX     j,d
004B  04002F          BR      for2
004E  00      endFor2: STOP
004F                  .END
```

Figure 6.35



Stack-indexed addressing

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

Local arrays

- Allocate total number of bytes of `v` with `SUBSP` and deallocate with `ADDSP`
- To access element `v[i]`, load `i` into index register, multiply by number of bytes per cell, use stack-indexed addressing (`sx`)

Figure 6.36

High-Order Language

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

int main () {
   int vector[4];
   int j;
   for (j = 0; j < 4; j++) {
      cin >> vector[j];
   }
   for (j = 3; j >= 0; j--) {
      cout << j  << ' ' << vector[j] << endl;
   }
   return 0;
}
```

Figure 6.36
(Continued)

Assembly Language

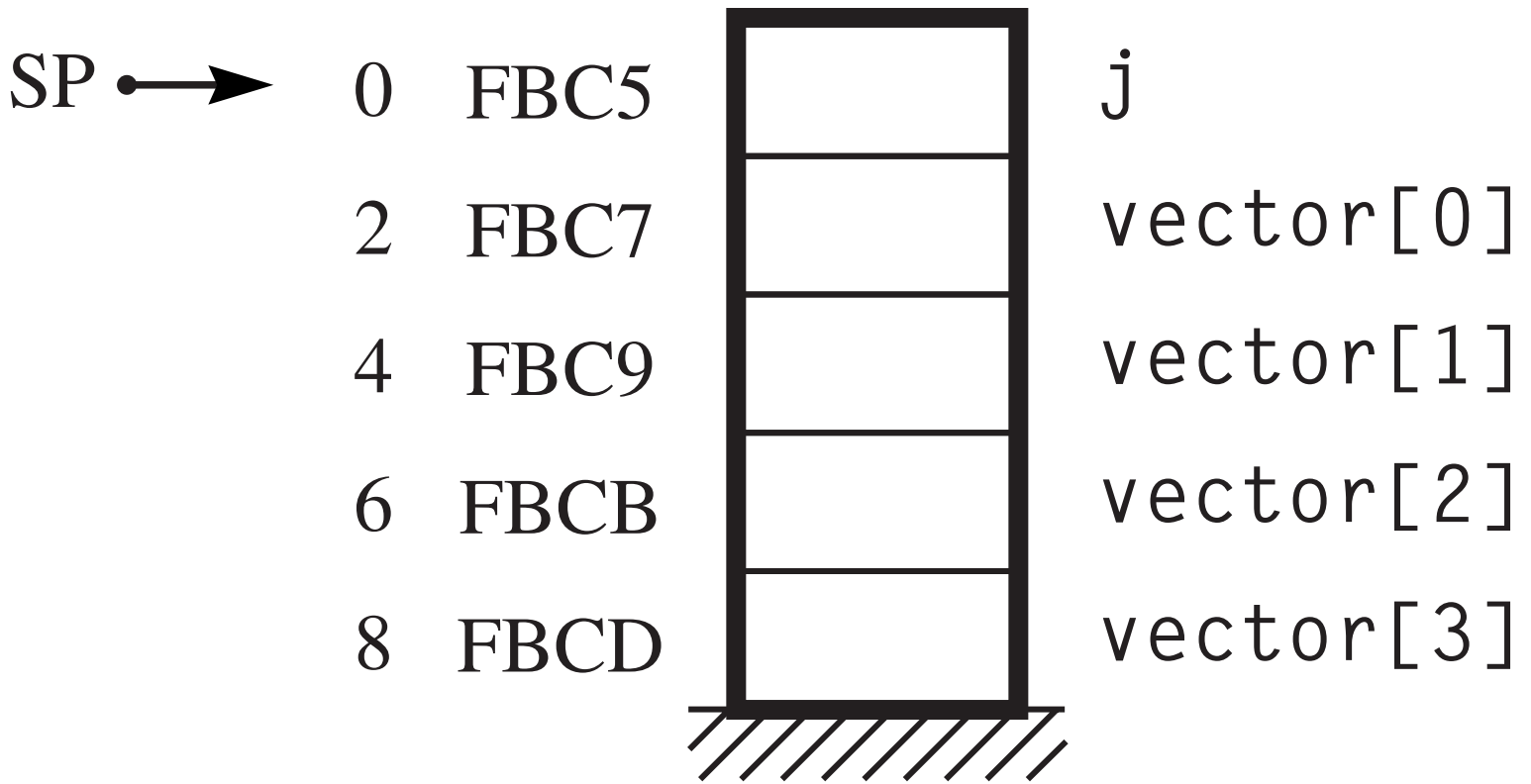
```
0000  040003          BR      main
                ;
                ;******* main ()
                vector: .EQUATE 2               ;local variable #2d4a
                j:      .EQUATE 0               ;local variable #2d
0003  68000A  main:     SUBSP   10,i            ;allocate #vector #j
0006  C80000            LDX     0,i             ;for (j = 0
0009  EB0000            STX     j,s
000C  B80004  for1:     CPX     4,i             ;    j < 4
000F  0E0022            BRGE    endFor1
0012  1D                ASLX                    ;    an integer is two bytes
0013  360002            DECI    vector,sx       ;    cin >> vector[j]
0016  CB0000            LDX     j,s             ;    j++)
0019  780001            ADDX    1,i
001C  EB0000            STX     j,s
001F  04000C            BR      for1
```

Figure 6.36
(Continued)

Assembly Language

```
0022  C80003 endFor1: LDX     3,i           ;for (j = 3
0025  EB0000          STX     j,s
0028  B80000 for2:    CPX     0,i           ;    j >= 0
002B  080047          BRLT    endFor2
002E  3B0000          DECO    j,s           ;    cout << j
0031  500020          CHARO   ' ',i         ;       << ' '
0034  1D              ASLX                  ;    an integer is two bytes
0035  3E0002          DECO    vector,sx     ;       << vector[j]
0038  50000A          CHARO   '\n',i        ;       << endl
003B  CB0000          LDX     j,s           ;    j--)
003E  880001          SUBX    1,i
0041  EB0000          STX     j,s
0044  040028          BR      for2
0047  60000A endFor2: ADDSP   10,i          ;deallocate #j #vector
004A  00              STOP
004B                  .END
```

Figure 6.37



Stack-indexed deferred addressing

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

Passing a local array as a parameter

- Push the address of the first element of the array `v` with `MOVSPA` followed by `ADDA` with immediate addressing
- To access element `v[i]`, load `i` into index register, multiply by number of bytes per cell, use stack-indexed deferred addressing (`sxf`)

Figure 6.38

High-Order Language

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

void getVect (int v[], int& n) {
   int j;
   cin >> n;
   for (j = 0; j < n; j++) {
      cin >> v[j];
   }
}

void putVect (int v[], int n) {
   int j;
   for (j = 0; j < n; j++) {
      cout << v[j] << ' ';
   }
   cout << endl;
}

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

Figure 6.38
(Continued)

Assembly Language

```
0000  04004C           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  680002 getVect: SUBSP   2,i             ;allocate #j
0006  340004           DECI    n,sf            ;cin >> n
0009  C80000           LDX     0,i             ;for (j = 0
000C  EB0000           STX     j,s
000F  BC0004 for1:     CPX     n,sf            ;   j < n
0012  0E0025           BRGE    endFor1
0015  1D               ASLX                    ;   an integer is two bytes
0016  370006           DECI    v,sxf           ;   cin >> v[j]
0019  CB0000           LDX     j,s             ;   j++)
001C  780001           ADDX    1,i
001F  EB0000           STX     j,s
0022  04000F           BR      for1
0025  5A     endFor1: RET2                    ;pop #j and retAddr
```

Figure 6.38
(Continued)

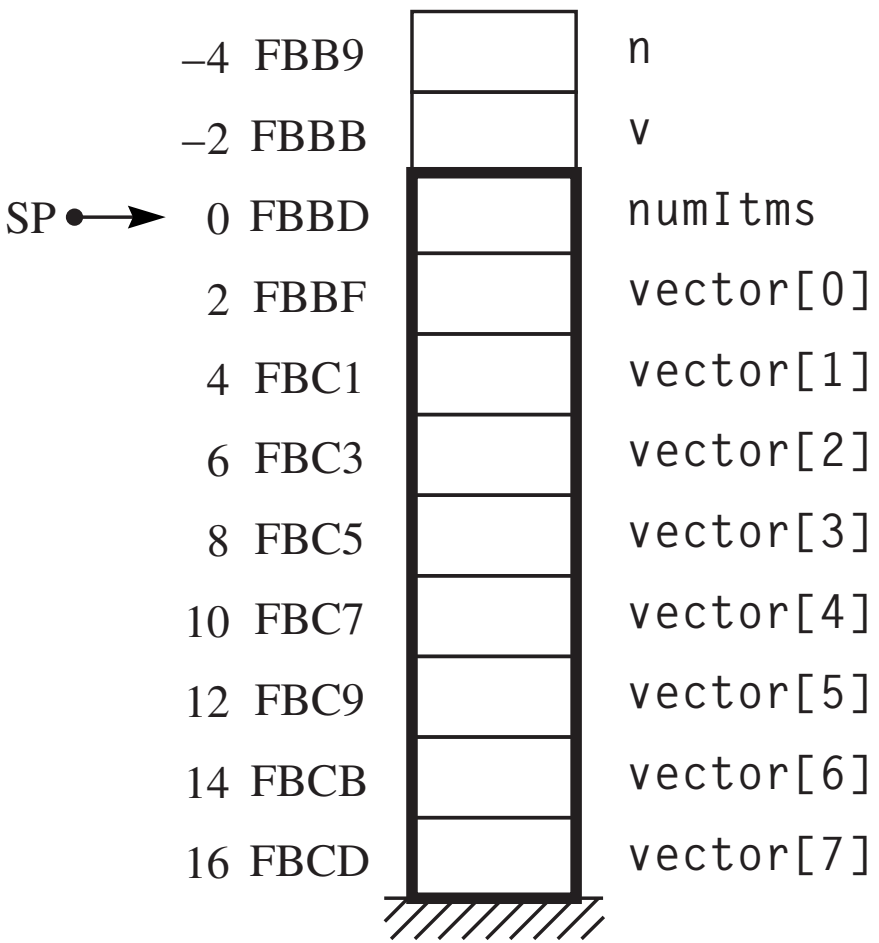
Assembly Language

```
                ;******* putVect (int v[], int n)
                v2:     .EQUATE 6               ;formal parameter #2h
                n2:     .EQUATE 4               ;formal parameter #2d
                j2:     .EQUATE 0               ;local variable #2d
0026  680002 putVect: SUBSP   2,i             ;allocate #j2
0029  C80000          LDX     0,i             ;for (j = 0
002C  EB0000          STX     j2,s
002F  BB0004 for2:    CPX     n2,s            ;   j < n
0032  0E0048          BRGE    endFor2
0035  1D              ASLX                    ;   an integer is two bytes
0036  3F0006          DECO    v2,sxf          ;   cout << v[j]
0039  500020          CHARO   ' ',i           ;      << ' '
003C  CB0000          LDX     j2,s            ;   i++)
003F  780001          ADDX    1,i
0042  EB0000          STX     j2,s
0045  04002F          BR      for2
0048  50000A endFor2: CHARO   '\n',i          ;cout << endl
004B  5A              RET2                    ;pop #j2 and retAddr
```

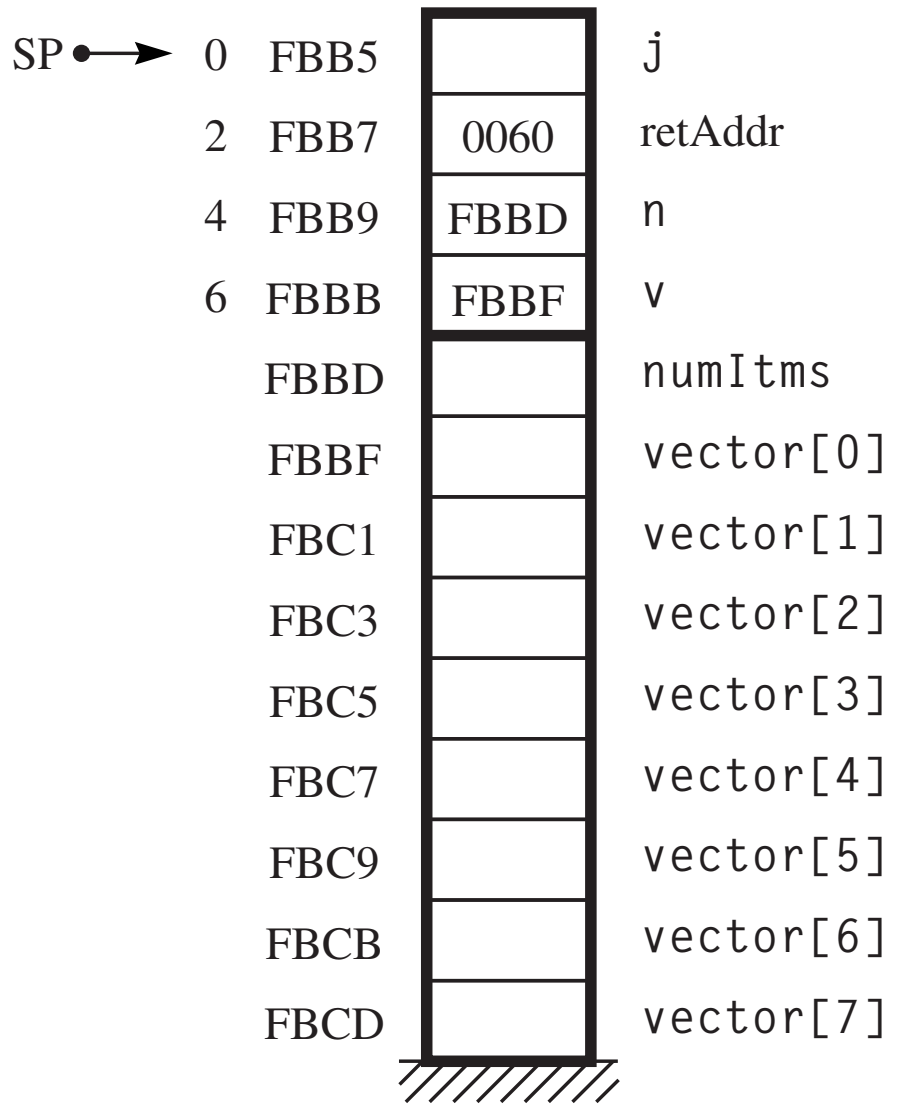
Figure 6.38
(Continued)

Assembly Language

```
                ;******* main ()
                vector:  .EQUATE 2              ;local variable #2d8a
                numItms: .EQUATE 0              ;local variable #2d
004C  680012    main:    SUBSP   18,i           ;allocate #vector #numItms
004F  02                 MOVSPA                 ;push address of vector
0050  700002             ADDA    vector,i
0053  E3FFFE             STA     -2,s
0056  02                 MOVSPA                 ;push address of numItms
0057  700000             ADDA    numItms,i
005A  E3FFFC             STA     -4,s
005D  680004             SUBSP   4,i            ;push #v #n
0060  160003             CALL    getVect        ;getVect (vector, numItms)
0063  600004             ADDSP   4,i            ;pop #n #v
0066  02                 MOVSPA                 ;push address of vector
0067  700002             ADDA    vector,i
006A  E3FFFE             STA     -2,s
006D  C30000             LDA     numItms,s      ;push value of numItms
0070  E3FFFC             STA     -4,s
0073  680004             SUBSP   4,i            ;push #v2 #n2
0076  160026             CALL    putVect        ;putVect (vector, numItms)
0079  600004             ADDSP   4,i            ;pop #n2 #v2
007C  600012             ADDSP   18,i           ;deallocate #numItms #vector
007F  00                 STOP
0080                     .END
```



(a) Before calling `getVect`.



(b) After calling `getVect`.

Passing a global array as a parameter

- Push the address of the first element of the array `v` with `LDA` with immediate addressing
- To access element `v[i]`, load `i` into index register, multiply by number of bytes per cell, use stack-indexed deferred addressing (`sxf`)

(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 `LDX` to load the index register with the switch value
- Execute `ASLX` once, because an address occupies two bytes
- Execute `BR` with indexed addressing

Figure 6.40

High-Order Language

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

int main () {
   int guess;
   cout << "Pick a number 0..3: ";
   cin >> guess;
   switch (guess) {
      case 0: cout << "Not close"; break;
      case 1: cout << "Close"; break;
      case 2: cout << "Right on"; break;
      case 3: cout << "Too high";
   }
   cout << endl;
   return 0;
}
```

Figure 6.40
(Continued)

Assembly Language

```
0000  040003          BR      main
                ;
                ;******* main ()
                guess:  .EQUATE 0               ;local variable #2d
0003  680002  main:     SUBSP   2,i             ;allocate #guess
0006  410037            STRO    msgIn,d         ;cout << "Pick a number 0..3: "
0009  330000            DECI    guess,s         ;cin >> Guess
000C  CB0000            LDX     guess,s         ;switch (Guess)
000F  1D                ASLX                    ;addresses occupy two bytes
0010  050013            BR      guessJT,x
0013  001B    guessJT:  .ADDRSS case0
0015  0021              .ADDRSS case1
0017  0027              .ADDRSS case2
0019  002D              .ADDRSS case3
001B  41004C  case0:    STRO    msg0,d          ;cout << "Not close"
001E  040030            BR      endCase         ;break
0021  410056  case1:    STRO    msg1,d          ;cout << "Close"
0024  040030            BR      endCase         ;break
0027  41005C  case2:    STRO    msg2,d          ;cout << "Right on"
002A  040030            BR      endCase         ;break
002D  410065  case3:    STRO    msg3,d          ;cout << "Too high"
0030  50000A  endCase:  CHARO   '\n',i          ;count << endl
0033  600002            ADDSP   2,i             ;deallocate #guess
0036  00                STOP
```

Figure 6.40
(Continued)

Assembly Language

```
0037   506963 msgIn:    .ASCII   "Pick a number 0..3: \x00"
       ...
004C   4E6F74 msg0:     .ASCII   "Not close\x00"
       ...
0056   436C6F msg1:     .ASCII   "Close\x00"
       ...
005C   526967 msg2:     .ASCII   "Right on\x00"
       ...
0065   546F6F msg3:     .ASCII   "Too high\x00"
       ...
006E                    .END
```

Global pointers

- Allocate storage for the pointer with `.BLOCK 2` because an address occupies two bytes
- Access the pointer with direct addressing (`d`)
- Access the cell to which the pointer points with indirect addressing (`n`)

Operator `new` calling protocol

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

Figure 6.41

High-Order Language

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

int *a, *b, *c;

int main () {
   a = new int;
   *a = 5;
   b = new int;
   *b = 3;
   c = a;
   a = b;
   *a = 2 + *c;
   cout << "*a = " << *a << endl;
   cout << "*b = " << *b << endl;
   cout << "*c = " << *c << endl;
   return 0;
}
```

Figure 6.41
(Continued)

Assembly Language

```
0000  040009          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:   LDA     2,i         ;a = new int
000C  16006A          CALL    new         ;#a
000F  E90003          STX     a,d
0012  C00005          LDA     5,i         ;*a = 5
0015  E20003          STA     a,n
0018  C00002          LDA     2,i         ;b = new int
001B  16006A          CALL    new         ;#b
001E  E90005          STX     b,d
0021  C00003          LDA     3,i         ;*b = 3
0024  E20005          STA     b,n
0027  C10003          LDA     a,d         ;c = a
002A  E10007          STA     c,d
002D  C10005          LDA     b,d         ;a = b
0030  E10003          STA     a,d
0033  C00002          LDA     2,i         ;*a = 2 + *c
0036  720007          ADDA    c,n
0039  E20003          STA     a,n
```

Figure 6.41
(Continued)

Assembly Language

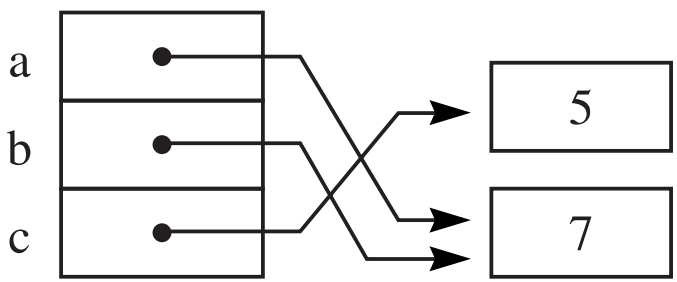
```
003C  410058           STRO    msg0,d          ;cout << "*a = "
003F  3A0003           DECO    a,n             ;   << *a
0042  50000A           CHARO   '\n',i          ;   << endl
0045  41005E           STRO    msg1,d          ;cout << "*b = "
0048  3A0005           DECO    b,n             ;   << *b
004B  50000A           CHARO   '\n',i          ;   << endl
004E  410064           STRO    msg2,d          ;cout << "*c = "
0051  3A0007           DECO    c,n             ;   << *c
0054  50000A           CHARO   '\n',i          ;   << endl
0057  00               STOP
0058  2A6120 msg0:     .ASCII  "*a = \x00"
      3D2000
005E  2A6220 msg1:     .ASCII  "*b = \x00"
      3D2000
0064  2A6320 msg2:     .ASCII  "*c = \x00"
      3D2000
```

Figure 6.41
(Continued)

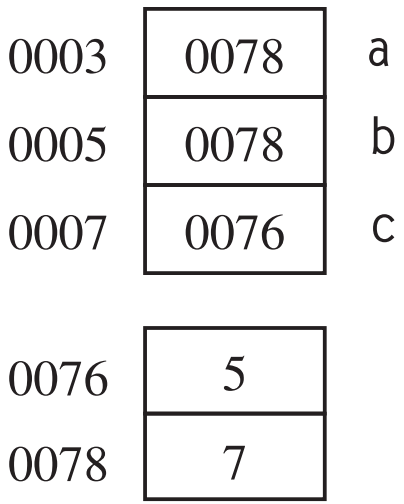
Assembly Language

```
                ;******* operator new
                ;       Precondition: A contains number of bytes
                ;       Postcondition: X contains pointer to bytes
006A   C90074 new:      LDX     hpPtr,d         ;returned pointer
006D   710074           ADDA    hpPtr,d         ;allocate from heap
0070   E10074           STA     hpPtr,d         ;update hpPtr
0073   58               RET0
0074   0076   hpPtr:    .ADDRSS heap            ;address of next free byte
0076   00     heap:     .BLOCK  1               ;first byte in the heap
0077                    .END
```

Figure 6.42



(a) Global pointers at level HOL6.



(b) The global pointers at level Asmb5.

Local pointers

- Allocate storage for the pointer with `SUBSP` and deallocate with `ADDSP`
- Access the pointer with stack-relative addressing (`s`)
- Access the cell to which the pointer points with stack-relative deferred addressing (`sf`)

Figure 6.43
(Continued)

High-Order Language

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

int main () {
   int *a, *b, *c;
   a = new int;
   *a = 5;
   b = new int;
   *b = 3;
   c = a;
   a = b;
   *a = 2 + *c;
   cout << "*a = " << *a << endl;
   cout << "*b = " << *b << endl;
   cout << "*c = " << *c << endl;
   return 0;
}
```

Figure 6.43
(Continued)

Assembly Language

```
0000  040003          BR      main
                ;
                ;******* main ()
                a:      .EQUATE 4               ;local variable #2h
                b:      .EQUATE 2               ;local variable #2h
                c:      .EQUATE 0               ;local variable #2h
0003  680006  main:   SUBSP   6,i             ;allocate #a #b #c
0006  C00002          LDA     2,i             ;a = new int
0009  16006A          CALL    new             ;#a
000C  EB0004          STX     a,s
000F  C00005          LDA     5,i             ;*a = 5
0012  E40004          STA     a,sf
0015  C00002          LDA     2,i             ;b = new int
0018  16006A          CALL    new             ;#b
001B  EB0002          STX     b,s
001E  C00003          LDA     3,i             ;*b = 3
0021  E40002          STA     b,sf
0024  C30004          LDA     a,s             ;c = a
0027  E30000          STA     c,s
002A  C30002          LDA     b,s             ;a = b
002D  E30004          STA     a,s
0030  C00002          LDA     2,i             ;*a = 2 + *c
0033  740000          ADDA    c,sf
0036  E40004          STA     a,sf
```

Figure 6.43
(Continued)

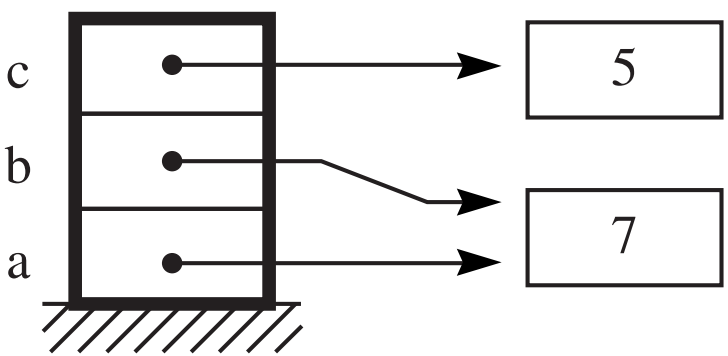
Assembly Language

```
0039  410058           STRO    msg0,d        ;cout << "*a = "
003C  3C0004           DECO    a,sf          ;   << *a
003F  50000A           CHARO   '\n',i        ;   << endl
0042  41005E           STRO    msg1,d        ;cout << "*b = "
0045  3C0002           DECO    b,sf          ;   << *b
0048  50000A           CHARO   '\n',i        ;   << endl
004B  410064           STRO    msg2,d        ;cout << "*c = "
004E  3C0000           DECO    c,sf          ;   << *c
0051  50000A           CHARO   '\n',i        ;   << endl
0054  600006           ADDSP   6,i           ;deallocate #c #b #a
0057  00               STOP
0058  2A6120 msg0:     .ASCII  "*a = \x00"
      3D2000
005E  2A6220 msg1:     .ASCII  "*b = \x00"
      3D2000
0064  2A6320 msg2:     .ASCII  "*c = \x00"
      3D2000
```

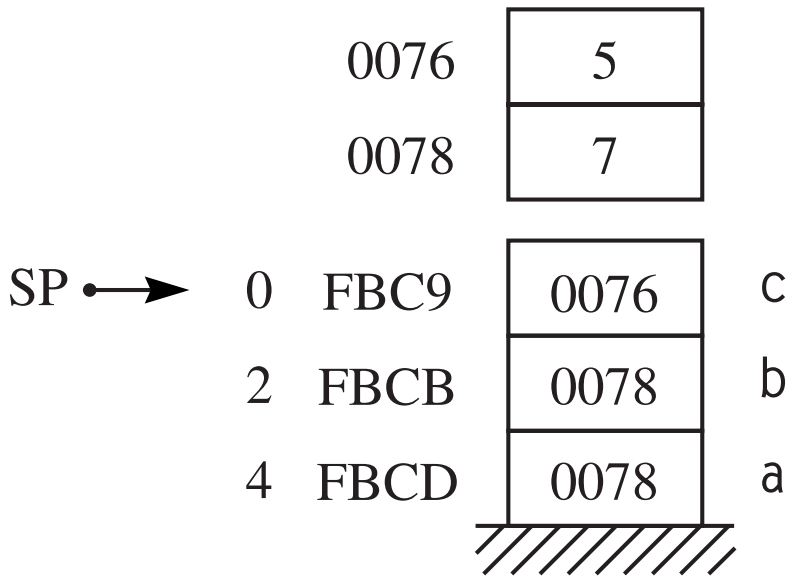
Figure 6.43
(Continued)

Assembly Language

```
                ;******* operator new
                ;       Precondition: A contains number of bytes
                ;       Postcondition: X contains pointer to bytes
006A   C90074 new:      LDX     hpPtr,d         ;returned pointer
006D   710074           ADDA    hpPtr,d         ;allocate from heap
0070   E10074           STA     hpPtr,d         ;update hpPtr
0073   58               RET0
0074   0076   hpPtr:    .ADDRSS heap            ;address of next free byte
0076   00     heap:     .BLOCK  1               ;first byte in the heap
0077                    .END
```



(a) Local pointers at level HOL6.



(b) The 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 the total number of bytes in the `struct` with `.BLOCK`
- To access a field, load the field into the index register with immediate addressing followed by an instruction with indexed addressing (`x`)

Figure 6.45

High-Order Language

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

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

int main () {
   cin >> bill.first >> bill.last >> bill.age >> bill.gender;
   cout << "Initials: " << bill.first << bill.last << endl;
   cout << "Age: " << bill.age << endl;
   cout << "Gender: ";
   if (bill.gender == 'm') {
      cout << "male\n";
   }
   else {
      cout << "female\n";
   }
   return 0;
}
```

Figure 6.45
(Continued)

Assembly Language

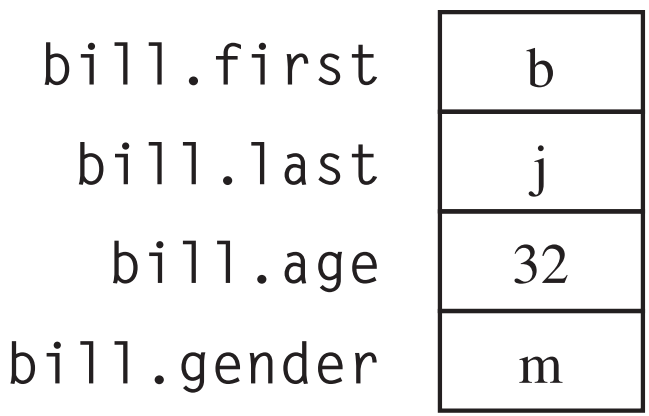
```
0000  040008          BR      main
              first:  .EQUATE 0           ;struct field #1c
              last:   .EQUATE 1           ;struct field #1c
              age:    .EQUATE 2           ;struct field #2d
              gender: .EQUATE 4           ;struct field #1c
0003  000000 bill:    .BLOCK  5           ;global variable #first #last #age
      0000
              ;
              ;******* main ()
0008  C80000 main:    LDX     first,i     ;cin >> bill.first
000B  4D0003          CHARI   bill,x
000E  C80001          LDX     last,i      ;   >>bill.last
0011  4D0003          CHARI   bill,x
0014  C80002          LDX     age,i       ;   >>bill.age
0017  350003          DECI    bill,x
001A  C80004          LDX     gender,i    ;   >>bill.gender
001D  4D0003          CHARI   bill,x
0020  41005A          STRO    msg0,d      ;cout << "Initials: "
0023  C80000          LDX     first,i     ;   << bill.first
0026  550003          CHARO   bill,x
0029  C80001          LDX     last,i      ;   << bill.last
002C  550003          CHARO   bill,x
002F  50000A          CHARO   '\n',i      ;   << endl
```

Figure 6.45
(Continued)

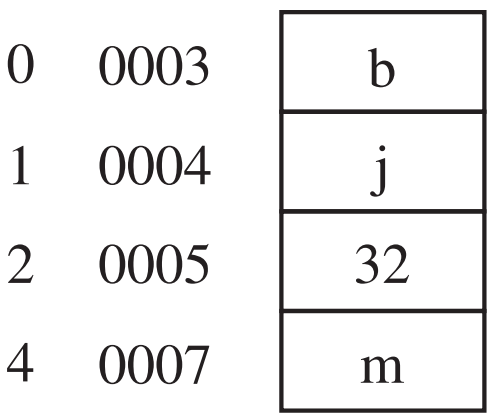
Assembly Language

```
0032  410065           STRO    msg1,d          ;cout << "Age: "
0035  C80002           LDX     age,i           ;   << bill.age
0038  3D0003           DECO    bill,x
003B  50000A           CHARO   '\n',i          ;   << endl;
003E  41006B           STRO    msg2,d          ;cout << "Gender: "
0041  C80004           LDX     gender,i        ;if (bill.gender == 'm')
0044  C00000           LDA     0,i
0047  D50003           LDBYTEA bill,x
004A  B0006D           CPA     'm',i
004D  0C0056           BRNE    else
0050  410074           STRO    msg3,d          ;   cout << "male\n"
0053  040059           BR      endIf
0056  41007A else:     STRO    msg4,d          ;   cout << "female\n"
0059  00     endIf:    STOP
005A  496E69 msg0:     .ASCII  "Initials: \x00"
      ...
0065  416765 msg1:     .ASCII  "Age: \x00"
      ...
006B  47656E msg2:     .ASCII  "Gender: \x00"
      ...
0074  6D616C msg3:     .ASCII  "male\n\x00"
      ...
007A  66656D msg4:     .ASCII  "female\n\x00"
      ...
```

Figure 6.46



(a) A global structure at level HOL6.



(b) The 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
- Load the offset into the index register
- Access the field of the node to which the pointer points using stack-indexed deferred addressing (sxf)

Figure 6.47

High-Order Language

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

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

int main () {
   node *first, *p;
   int value;
   first = 0;
   cin >> value;
   while (value != -9999) {
      p = first;
      first = new node;
      first->data = value;
      first->next = p;
      cin >> value;
   }
   for (p = first; p != 0; p = p->next) {
      cout << p->data << ' ';
   }
   return 0;
```

Figure 6.47
(Continued)

Assembly Language

```
0000  040003          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  680006  main:   SUBSP   6,i             ;allocate #first #p #value
0006  C00000          LDA     0,i             ;first = 0
0009  E30004          STA     first,s
000C  330000          DECI    value,s         ;cin >> value
000F  C30000  while:  LDA     value,s         ;while (value != -9999)
0012  B0D8F1          CPA     -9999,i
0015  0A003F          BREQ    endWh
0018  C30004          LDA     first,s         ;   p = first
001B  E30002          STA     p,s
001E  C00004          LDA     4,i             ;   first = new node
0021  160067          CALL    new             ;   allocate #data #next
0024  EB0004          STX     first,s
0027  C30000          LDA     value,s         ;   first->data = value
002A  C80000          LDX     data,i
002D  E70004          STA     first,sxf
```

Figure 6.47
(Continued)

Assembly Language

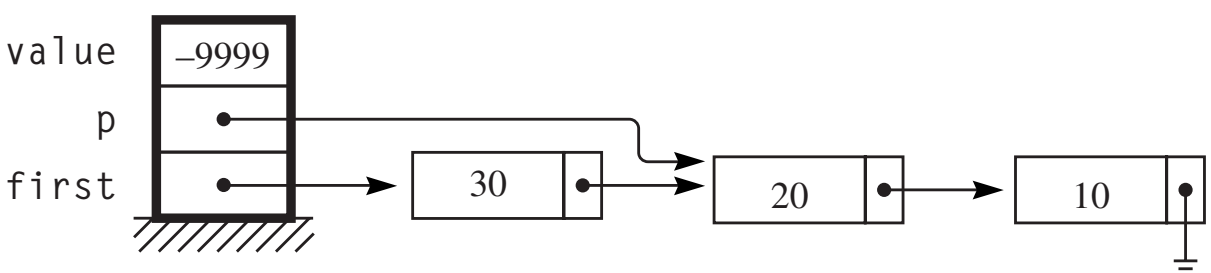
```
0030  C30002          LDA     p,s             ;   first->next = p
0033  C80002          LDX     next,i
0036  E70004          STA     first,sxf
0039  330000          DECI    value,s         ;   cin >> value
003C  04000F          BR      while
003F  C30004 endWh:   LDA     first,s         ;for (p = first
0042  E30002          STA     p,s
0045  C30002 for:     LDA     p,s             ;   p != 0
0048  B00000          CPA     0,i
004B  0A0063          BREQ    endFor
004E  C80000          LDX     data,i          ;   cout << p->data
0051  3F0002          DECO    p,sxf
0054  500020          CHARO   ' ',i           ;      << ' '
0057  C80002          LDX     next,i          ;   p = p->next)
005A  C70002          LDA     p,sxf
005D  E30002          STA     p,s
0060  040045          BR      for
0063  600006 endFor:  ADDSP   6,i             ;deallocate #value #p #first
0066  00              STOP
```

Figure 6.47
(Continued)

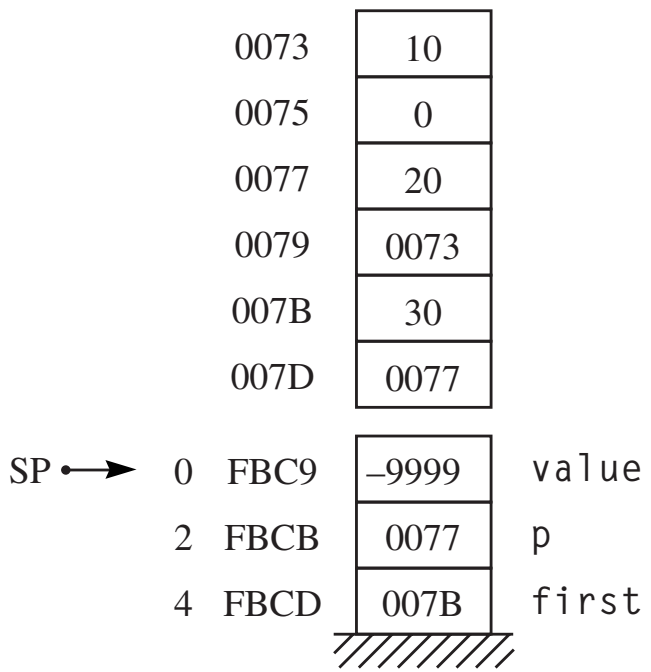
Assembly Language

```
                ;******* operator new
                ;       Precondition: A contains number of bytes
                ;       Postcondition: X contains pointer to bytes
0067   C90071 new:    LDX     hpPtr,d         ;returned pointer
006A   710071          ADDA    hpPtr,d         ;allocate from heap
006D   E10071          STA     hpPtr,d         ;update hpPtr
0070   58              RET0
0071   0073   hpPtr:  .ADDRSS heap            ;address of next free byte
0073   00     heap:   .BLOCK  1               ;first byte in the heap
0074                   .END
```

Figure 6.48



(a) The linked list at level HOL6.



(b) The linked list at level Asmb5.

Linked data structure with a global pointer

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

Figure 6.49