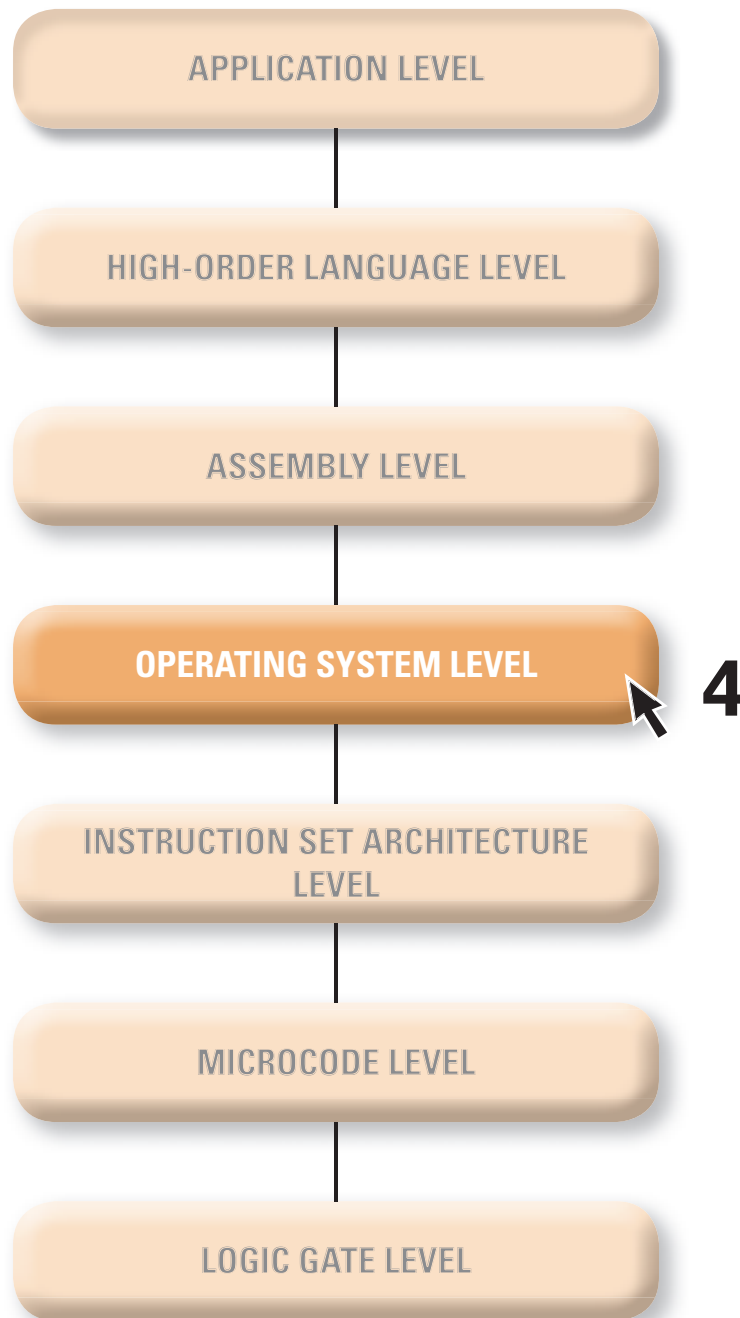


Operating System

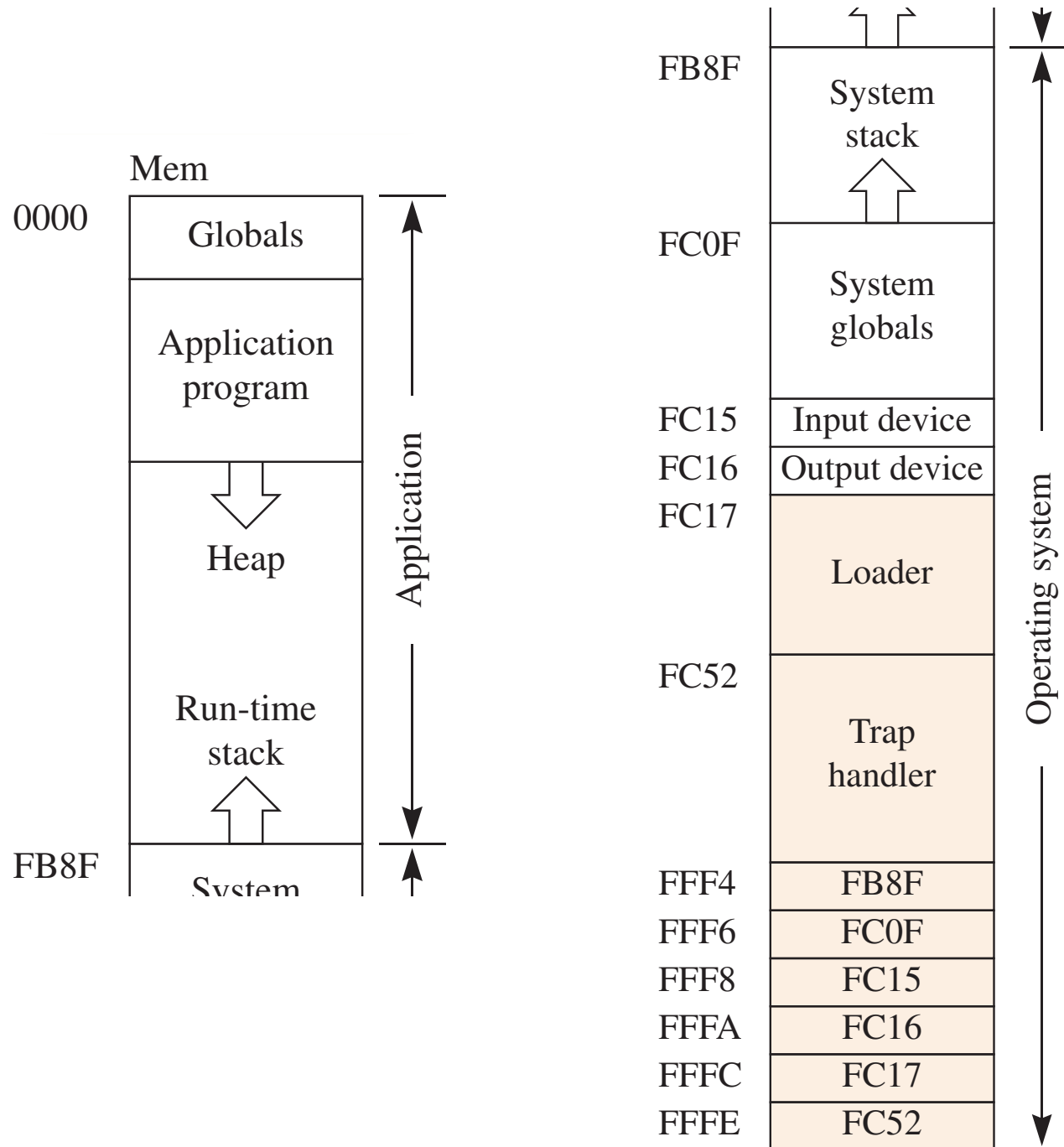


Three types of operating systems

- Single-user
 - ▶ Hand-held devices, e.g. smart phones
- Multi-user
 - ▶ Desktop and laptop computers
- Real-time
 - ▶ Equipment control, e.g. car engine

The Pep/9 OS

- An operating system is a program
- One function of an operating system is to manage the jobs (application programs) that users submit
- Because the operating system is itself a program, it is stored in memory
- The location of the OS program relative to the application programs is the *memory map*



The Pep/9 loader

- The purpose is to load the application program into memory starting at address 0000

- When you invoke the Pep/9 loader:

$SP \leftarrow \text{Mem}[\text{FFF6}]$

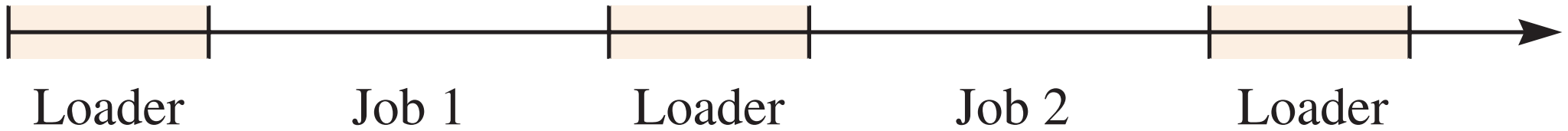
$PC \leftarrow \text{Mem}[\text{FFFC}]$

The .BURN command

- If .BURN is in a program, the assembler assumes the program will be burned into ROM
- It generates code for instructions that follow the .BURN
- It does *not* generate code for instructions that precede the .BURN
- It computes symbol values assuming the operand of .BURN is the *last* address

Program termination

- Pep/9 OS
 - ▶ When a program terminates with STOP, control returns to the user of the Pep/9 simulator
- Real-world OS
 - ▶ When a program terminates, the computer does not stop, but returns control to the operating system

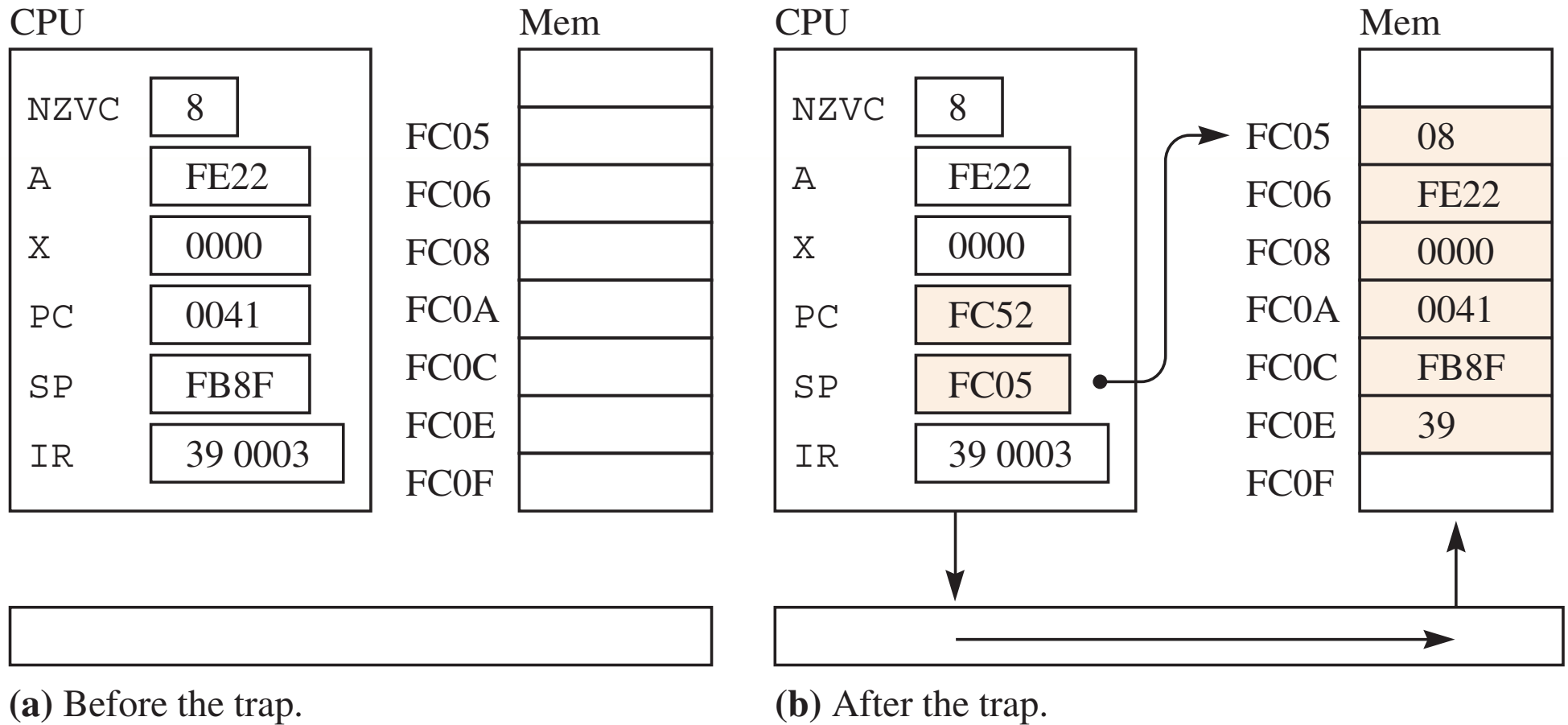


Traps

- Executed by the unimplemented nonunary instructions DECI, DECO, HEXO, STRO, NOP and the unary instructions NOP0, NOP1
- Similar to the CALL instruction, but all the registers, not just SP, are stored on the system stack

The trap mechanism

Temp	\leftarrow Mem[FFF6] ;
Mem[Temp - 1]	\leftarrow IR<0..7> ;
Mem[Temp - 3]	\leftarrow SP ;
Mem[Temp - 5]	\leftarrow PC ;
Mem[Temp - 7]	\leftarrow X ;
Mem[Temp - 9]	\leftarrow A ;
Mem[Temp - 10]<4..7>	\leftarrow NZVC ;
SP	\leftarrow Temp - 10 ;
PC	\leftarrow Mem[FFFE]



Processes

- Process
 - ▶ A program during execution
- Process Control Block
 - ▶ The block of information in main memory that contains a copy of the trapped processes' registers

The return from trap instruction

- Instruction specifier: 0000 0010
- Mnemonic: RETR

NZVC \leftarrow Mem[SP]⟨4..7⟩ ;

A \leftarrow Mem[SP + 1] ;

X \leftarrow Mem[SP + 3] ;

PC \leftarrow Mem[SP + 5] ;

SP \leftarrow Mem[SP + 7]

The trap handlers

0010 011n	NOPn	Unary no-operation
0010 1aaa	NOP	Nonunary no-operation
0011 0aaa	DECI	Nonunary decimal input
0011 1aaa	DECO	Nonunary decimal output
0100 0aaa	HEXO	Nonunary hex output
0100 1aaa	STRO	Nonunary string output

The test for NOPn

0010 0110 NOP0 Right-most bit 0

0010 0111 NOP1 Right-most bit 1

```

;***** Trap handler
oldIR:  .EQUATE 9                ;Stack address of IR on trap
;
FC52 DB0009 trap:  LDBX      oldIR,s    ;X <- trapped IR
FC55 B80028      CPBX      0x0028,i    ;If X >= first nonunary trap opcode
FC58 1CFC67      BRGE      nonUnary    ; trap opcode is nonunary
;
FC5B 880001 unary:  ANDX      0x0001,i  ;Mask out all but rightmost bit
FC5E 0B          ASLX              ;Two bytes per address
FC5F 25FC63      CALL      unaryJT,x    ;Call unary trap routine
FC62 02          RETTR              ;Return from trap
;
FC63 FD6B unaryJT: .ADDRSS opcode26    ;Address of NOP0 subroutine
FC65 FD6C          .ADDRSS opcode27    ;Address of NOP1 subroutine

```


The test for the nonunary trap instructions

0 if the trap IR contains 0010 1aaa NOP

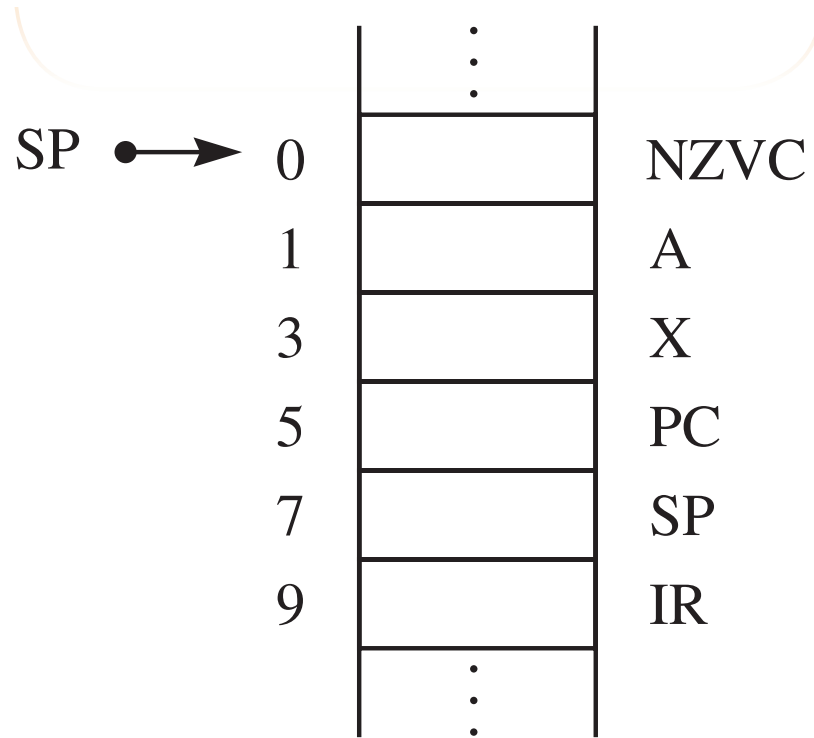
1 if the trap IR contains 0011 0aaa DECI

2 if the trap IR contains 0011 1aaa DECO

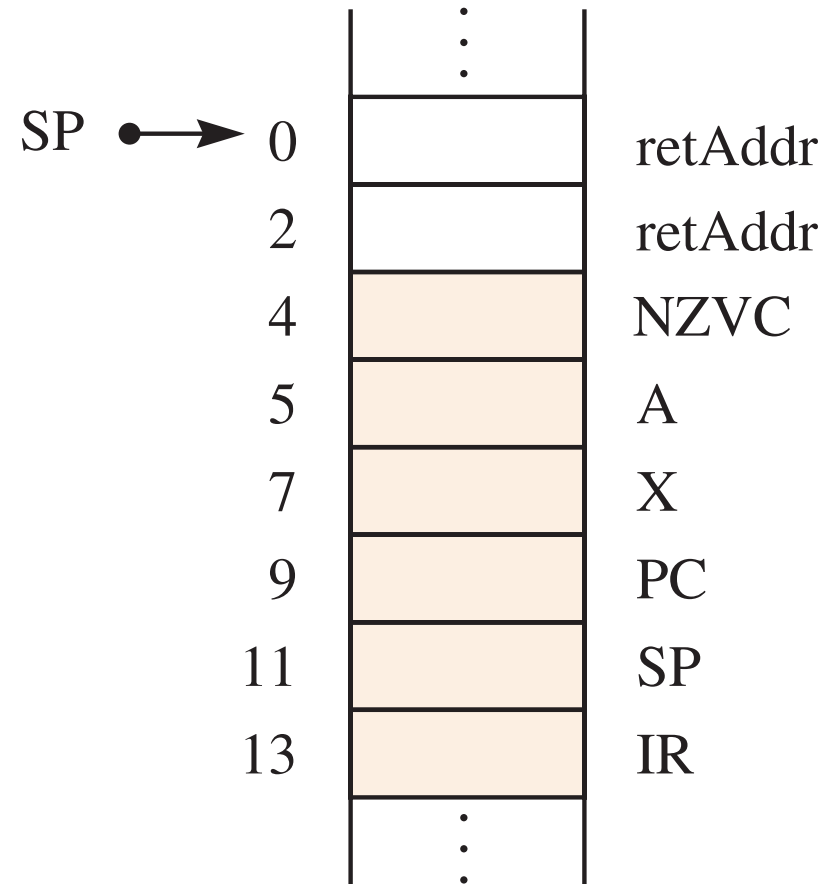
3 if the trap IR contains 0100 0aaa HEXO

4 if the trap IR contains 0100 1aaa STRO

FC67	0D	nonUnary:ASRX		;Trap opcode is nonunary
FC68	0D	ASRX		;Discard addressing mode bits
FC69	0D	ASRX		
FC6A	780005	SUBX	5,i	;Adjust so that NOP opcode = 0
FC6D	0B	ASLX		;Two bytes per address
FC6E	25FC72	CALL	nonUnJT,x	;Call nonunary trap routine
FC71	02	return: RETTR		;Return from trap
		;		
FC72	FD6D	nonUnJT: .ADDRSS	opcode28	;Address of NOP subroutine
FC74	FD77	.ADDRSS	opcode30	;Address of DECI subroutine
FC76	FEEB	.ADDRSS	opcode38	;Address of DECO subroutine
FC78	FF76	.ADDRSS	opcode40	;Address of HEXO subroutine
FC7A	FFC2	.ADDRSS	opcode48	;Address of STRO subroutine



(a) Immediately after a trap.

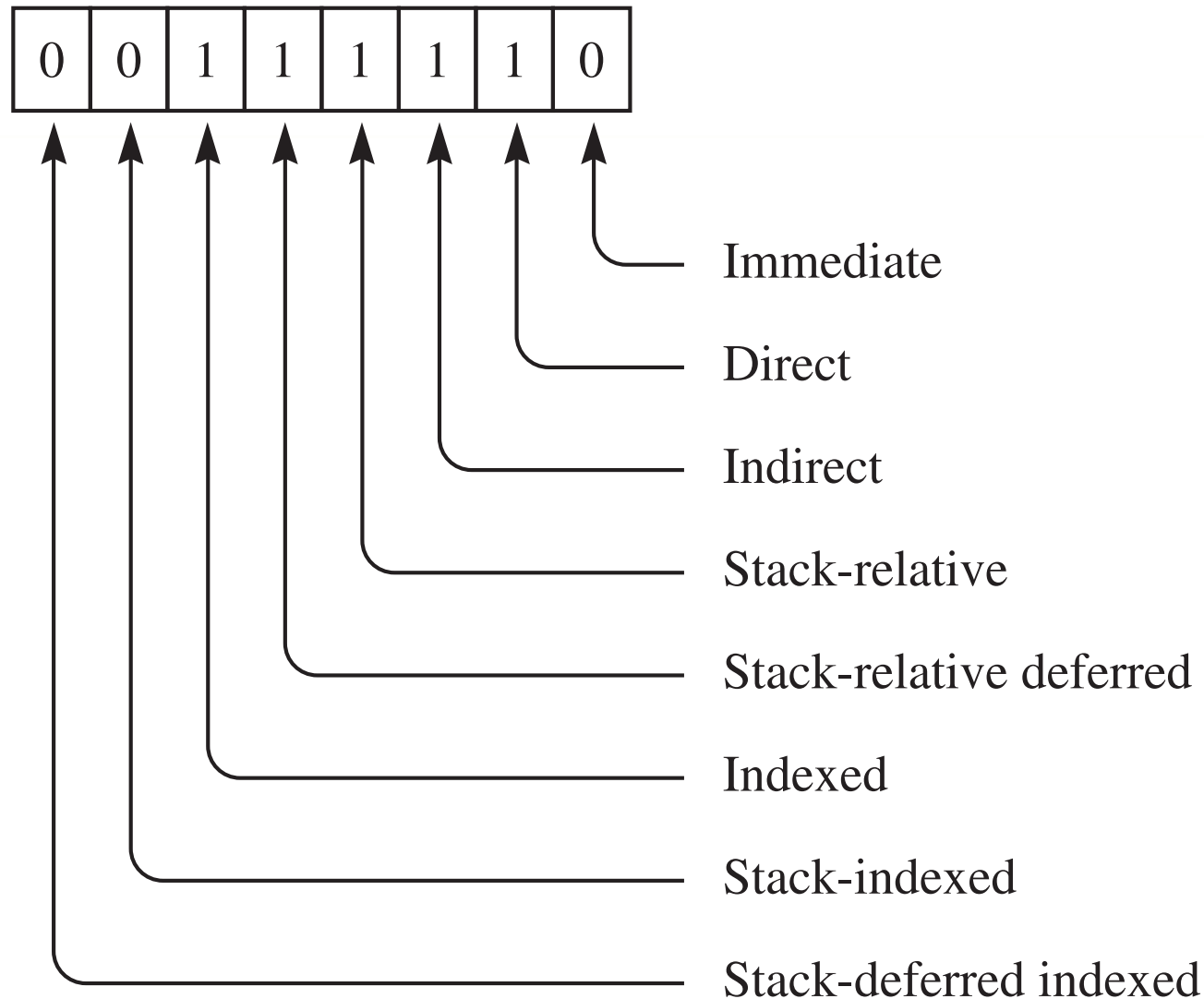


(b) With two return addresses on the run-time stack. The shaded region is the PCB.

Trap addressing mode assertion

- Precondition
 - ▶ `addrMask` is a bit mask representation of the set of allowable addressing modes, and the PCB of the stack instruction is on the system stack

Addressing modes for STRO



Trap addressing mode assertion

- Postcondition
 - ▶ If the addressing mode of the trap instruction is in the set of allowable addressing modes, control is returned to the trap handler. Otherwise, an invalid addressing mode message is output and the program halts with a fatal run-time error.

```

;***** Assert valid trap addressing mode
oldIR4:  .EQUATE 13                ;oldIR + 4 with two return addresses
FC7C  D00001 assertAd:LDBA      1,i          ;A <- 1
FC7F  DB000D                LDBX      oldIR4,s    ;X <- OldIR
FC82  880007                ANDX      0x0007,i    ;Keep only the addressing mode bits
FC85  18FC8F                BREQ      testAd      ;000 = immediate addressing
FC88  0A      loop:        ASLA                ;Shift the 1 bit left
FC89  780001                SUBX      1,i          ;Subtract from addressing mode count
FC8C  1AFC88                BRNE      loop        ;Try next addressing mode
FC8F  81FC11 testAd:        ANDA      addrMask,d   ;AND the 1 bit with legal modes
FC92  18FC96                BREQ      addrErr
FC95  01                  RET                  ;Legal addressing mode, return
FC96  D0000A addrErr:      LDBA      '\n',i
FC99  F1FC16                STBA      charOut,d
FC9C  C0FCA9                LDWA      trapMsg,i    ;Push address of error message
FC9F  E3FFFE                STWA      -2,s
FCA2  580002                SUBSP     2,i          ;Call print subroutine
FCA5  24FFDE                CALL      prntMsg
FCA8  00                  STOP                ;Halt: Fatal runtime error
FCA9  455252 trapMsg:      .ASCII    "ERROR: Invalid trap addressing mode.\x00"
...

```

Trap operand address computation

- Precondition
 - ▶ The PCB of the stack instruction is on the system stack
- Postcondition
 - ▶ opAddr contains the address of the operand according to the addressing mode of the trap instruction


```

;***** Set address of trap operand
oldX4:    .EQUATE 7           ;oldX + 4 with two return addresses
oldPC4:   .EQUATE 9           ;oldPC + 4 with two return addresses
oldSP4:   .EQUATE 11          ;oldSP + 4 with two return addresses
FCCE DB000D setAddr: LDBX      oldIR4,s      ;X <- old instruction register
FCD1 880007      ANDX      0x0007,i        ;Keep only the addressing mode bits
FCD4 0B          ASLX                      ;Two bytes per address
FCD5 13FCD8      BR        addrJT,x
FCD8 FCE8      addrJT:  .ADDRSS addrI        ;Immediate addressing
FCDA FCF2      .ADDRSS addrD        ;Direct addressing
FCDC FCFF      .ADDRSS addrN        ;Indirect addressing
FCDE FD0F      .ADDRSS addrS        ;Stack-relative addressing
FCE0 FD1F      .ADDRSS addrSF       ;Stack-relative deferred addressing
FCE2 FD32      .ADDRSS addrX        ;Indexed addressing
FCE4 FD42      .ADDRSS addrSX       ;Stack-indexed addressing
FCE6 FD55      .ADDRSS addrSFX      ;Stack-deferred indexed addressing

;
FCE8 CB0009      addrI:  LDWX      oldPC4,s    ;Immediate addressing
FCEB 780002      SUBX      2,i        ;Oprnd = OprndSpec
FCEE E9FC13      STWX      opAddr,d
FCF1 01          RET

```

FCF2	CB0009	addrD:	LDWX	oldPC4,s	;Direct addressing
FCF5	780002		SUBX	2,i	;Oprnd = Mem[OprndSpec]
FCF8	CD0000		LDWX	0,x	
FCFB	E9FC13		STWX	opAddr,d	
FCFE	01		RET		
		;			
FCFF	CB0009	addrN:	LDWX	oldPC4,s	;Indirect addressing
FD02	780002		SUBX	2,i	;Oprnd = Mem[Mem[OprndSpec]]
FD05	CD0000		LDWX	0,x	
FD08	CD0000		LDWX	0,x	
FD0B	E9FC13		STWX	opAddr,d	
FD0E	01		RET		
		;			
FD0F	CB0009	addrS:	LDWX	oldPC4,s	;Stack-relative addressing
FD12	780002		SUBX	2,i	;Oprnd = Mem[SP + OprndSpec]
FD15	CD0000		LDWX	0,x	
FD18	6B000B		ADDX	oldSP4,s	
FD1B	E9FC13		STWX	opAddr,d	
FD1E	01		RET		

FD1F	CB0009	addrSF:	LDWX	oldPC4,s	;Stack-relative deferred addressing
FD22	780002		SUBX	2,i	;Oprnd = Mem[Mem[SP + OprndSpec]]
FD25	CD0000		LDWX	0,x	
FD28	6B000B		ADDX	oldSP4,s	
FD2B	CD0000		LDWX	0,x	
FD2E	E9FC13		STWX	opAddr,d	
FD31	01		RET		
		;			
FD32	CB0009	addrX:	LDWX	oldPC4,s	;Indexed addressing
FD35	780002		SUBX	2,i	;Oprnd = Mem[OprndSpec + X]
FD38	CD0000		LDWX	0,x	
FD3B	6B0007		ADDX	oldX4,s	
FD3E	E9FC13		STWX	opAddr,d	
FD41	01		RET		

FD42	CB0009	addrSX:	LDWX	oldPC4,s	;Stack-indexed addressing
FD45	780002		SUBX	2,i	;Oprnd = Mem[SP + OprndSpec + X]
FD48	CD0000		LDWX	0,x	
FD4B	6B0007		ADDX	oldX4,s	
FD4E	6B000B		ADDX	oldSP4,s	
FD51	E9FC13		STWX	opAddr,d	
FD54	01		RET		
		;			
FD55	CB0009	addrSFX:	LDWX	oldPC4,s	;Stack-deferred indexed addressing
FD58	780002		SUBX	2,i	;Oprnd = Mem[Mem[SP + OprndSpec] + X]
FD5B	CD0000		LDWX	0,x	
FD5E	6B000B		ADDX	oldSP4,s	
FD61	CD0000		LDWX	0,x	
FD64	6B0007		ADDX	oldX4,s	
FD67	E9FC13		STWX	opAddr,d	
FD6A	01		RET		

The no-operation trap handlers

- Do nothing when executed
- Provided for systems programmer to write her own trap handler

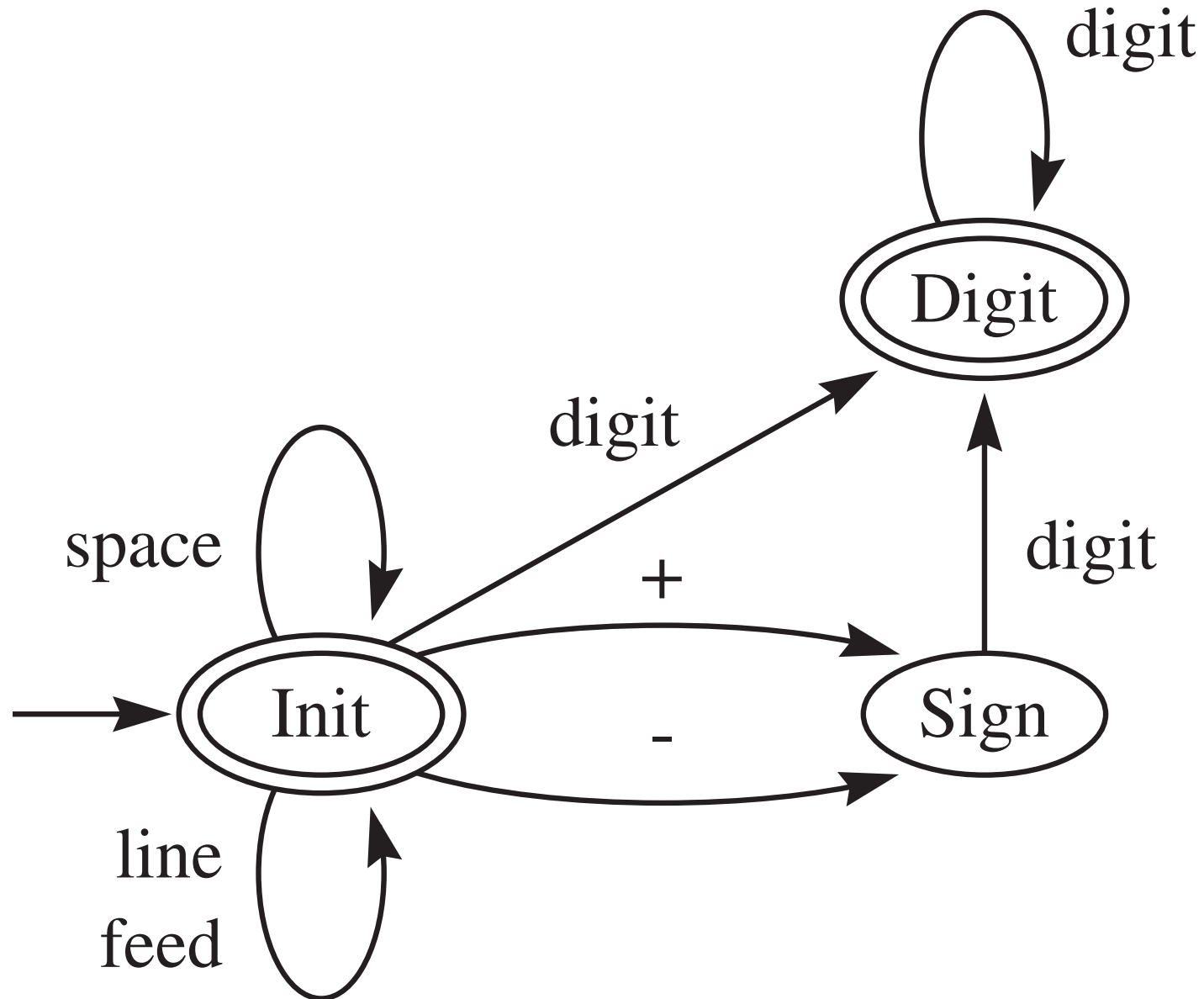
```

;***** Opcode 0x26
;The NOP0 instruction.
FD6B  01  opcode26:RET
;
;***** Opcode 0x27
;The NOP1 instruction.
FD6C  01  opcode27:RET
;
;***** Opcode 0x28
;The NOP instruction.
FD6D  C00001 opcode28:LDWA    0x0001,i    ;Assert i
FD70  E1FC11      STWA      addrMask,d
FD73  24FC7C      CALL      assertAd
FD76  01          RET

```

The DECI trap handler

- Parses the input, converting the string of ASCII characters to the proper bits in two's complement representation
- Based on a finite state machine




```
isOvfl := false
state := init
do
  CHARI asciiCh
  switch state
  case init:
    if (asciiCh == '+') {
      isNeg ← false
      state ← sign
    }
    else if (asciiCh == '-') {
      isNeg ← true
      state ← sign
    }
    else if (asciiCh is a Digit) {
      isNeg ← false
      total ← Value (asciiCh)
      state ← digit
    }
    else if (asciiCh is not <SPACE> or <LF>) {
      Exit with DECI error
    }
```

```
case sign:
    if (asciiCh is a Digit) {
        total ← Value(asciiCh)
        state ← digit
    }
    else {
        Exit with DECI error
    }
case digit:
    if (asciiCh is a Digit) {
        total ← 10 * total + Value(asciiCh)
        if (overflow) {
            isOvfl ← true
        }
    }
    else {
        Exit normally
    }
end switch
while (not exit)
```

```
;***** Opcode 0x30
;The DECI instruction.
;Input format: Any number of leading spaces or line feeds are
;allowed, followed by '+', '-' or a digit as the first character,
;after which digits are input until the first nondigit is
;encountered. The status flags N,Z and V are set appropriately
;by this DECI routine. The C status flag is not affected.
;
oldNZVC: .EQUATE 15           ;Stack address of NZVC on interrupt
;
total:    .EQUATE 11          ;Cumulative total of DECI number
asciiCh:  .EQUATE 10          ;asciiCh, one byte
valAscii: .EQUATE 8           ;value(asciiCh)
isOvfl:   .EQUATE 6           ;Overflow boolean
isNeg:    .EQUATE 4           ;Negative boolean
state:    .EQUATE 2           ;State variable
temp:     .EQUATE 0
;
init:     .EQUATE 0           ;Enumerated values for state
sign:     .EQUATE 1
digit:    .EQUATE 2
```

```

FD77  C000FE opcode30:LDWA    0x00FE,i    ;Assert d, n, s, sf, x, sx, sfx
FD7A  E1FC11          STWA    addrMask,d
FD7D  24FC7C          CALL    assertAd
FD80  24FCCE          CALL    setAddr    ;Set address of trap operand
FD83  58000D          SUBSP    13,i      ;Allocate storage for locals
FD86  C00000          LDWA     FALSE,i    ;isOvfl <- FALSE
FD89  E30006          STWA     isOvfl,s
FD8C  C00000          LDWA     init,i     ;state <- init
FD8F  E30002          STWA     state,s

;
FD92  D1FC15 do:      LDBA     charIn,d    ;Get asciiCh
FD95  F3000A          STBA     asciiCh,s
FD98  80000F          ANDA     0x000F,i    ;Set value(asciiCh)
FD9B  E30008          STWA     valAscii,s
FD9E  D3000A          LDBA     asciiCh,s   ;A<low> = asciiCh throughout the loop
FDA1  CB0002          LDWX     state,s     ;switch (state)
FDA4  0B              ASLX                     ;Two bytes per address
FDA5  13FDA8          BR      stateJT,x

;
FDA8  FDAE      stateJT: .ADDRSS sInit
FDAA  FE08          .ADDRSS sSign
FDAC  FE23          .ADDRSS sDigit

```

```

FDAE  B0002B sInit:  CPBA    '+' ,i      ;if (asciiCh == '+')
FDB1  1AFDC3          BRNE    ifMinus
FDB4  C80000          LDWX    FALSE,i      ;isNeg <- FALSE
FDB7  EB0004          STWX    isNeg,s
FDBA  C80001          LDWX    sign,i       ;state <- sign
FDBD  EB0002          STWX    state,s
FDC0  12FD92          BR      do

;

FDC3  B0002D ifMinus: CPBA    '-' ,i      ;else if (asciiCh == '-')
FDC6  1AFDD8          BRNE    ifDigit
FDC9  C80001          LDWX    TRUE,i       ;isNeg <- TRUE
FDCC  EB0004          STWX    isNeg,s
FDCF  C80001          LDWX    sign,i       ;state <- sign
FDD2  EB0002          STWX    state,s
FDD5  12FD92          BR      do

```

```

FDD8  B00030  ifDigit:  CPBA      '0',i          ;else if (asciiCh is a digit)
Fddb  16FDF9          BRLT      ifWhite
FDDE  B00039          CPBA      '9',i
FDE1  1EFDF9          BRGT      ifWhite
FDE4  C80000          LDWX      FALSE,i          ;isNeg <- FALSE
FDE7  EB0004          STWX      isNeg,s
FDEA  CB0008          LDWX      valAscii,s      ;total <- value(asciiCh)
FDED  EB000B          STWX      total,s
FDF0  C80002          LDWX      digit,i          ;state <- digit
FDF3  EB0002          STWX      state,s
FDF6  12FD92          BR        do

;

FDF9  B00020  ifWhite:  CPBA      ' ',i          ;else if (asciiCh is not a space
FDFC  18FD92          BREQ      do
FDFF  B0000A          CPBA      '\n',i          ;or line feed)
FE02  1AFEBE          BRNE      deciErr          ;exit with DECI error
FE05  12FD92          BR        do
    
```

```

FE08  B00030  sSign:  CPBA    '0',i          ;if asciiCh (is not a digit)
FE0B  16FEBE          BRLT    deciErr
FE0E  B00039          CPBA    '9',i
FE11  1EFEBE          BRGT    deciErr          ;exit with DECI error
FE14  CB0008          LDWX    valAscii,s      ;else total <- value(asciiCh)
FE17  EB000B          STWX    total,s
FE1A  C80002          LDWX    digit,i          ;state <- digit
FE1D  EB0002          STWX    state,s
FE20  12FD92          BR      do

;

FE23  B00030  sDigit:  CPBA    '0',i          ;if (asciiCh is not a digit)
FE26  16FE74          BRLT    deciNorm
FE29  B00039          CPBA    '9',i
FE2C  1EFE74          BRGT    deciNorm          ;exit normaly
FE2F  C80001          LDWX    TRUE,i          ;else X <- TRUE for later assignments
FE32  C3000B          LDWA    total,s          ;Multiply total by 10 as follows:
FE35  0A              ASLA
FE36  20FE3C          BRV     ovfl1          ;If overflow then
FE39  12FE3F          BR      L1

```

FE3C	EB0006	ovfl1:	STWX	isOvfl,s	;isOvfl <- TRUE
FE3F	E30000	L1:	STWA	temp,s	;Save 2 * total in temp
FE42	0A		ASLA		;Now, 4 * total
FE43	20FE49		BRV	ovfl2	;If overflow then
FE46	12FE4C		BR	L2	
FE49	EB0006	ovfl2:	STWX	isOvfl,s	;isOvfl <- TRUE
FE4C	0A	L2:	ASLA		;Now, 8 * total
FE4D	20FE53		BRV	ovfl3	;If overflow then
FE50	12FE56		BR	L3	
FE53	EB0006	ovfl3:	STWX	isOvfl,s	;isOvfl <- TRUE
FE56	630000	L3:	ADDA	temp,s	;Finally, 8 * total + 2 * total
FE59	20FE5F		BRV	ovfl4	;If overflow then
FE5C	12FE62		BR	L4	
FE5F	EB0006	ovfl4:	STWX	isOvfl,s	;isOvfl <- TRUE
FE62	630008	L4:	ADDA	valAscii,s	;A <- 10 * total + valAscii
FE65	20FE6B		BRV	ovfl5	;If overflow then
FE68	12FE6E		BR	L5	
FE6B	EB0006	ovfl5:	STWX	isOvfl,s	;isOvfl <- TRUE
FE6E	E3000B	L5:	STWA	total,s	;Update total
FE71	12FD92		BR	do	


```

FE74 C30004 deciNorm:LDWA isNeg,s ;If isNeg then
FE77 18FE90 BREQ setNZ
FE7A C3000B LDWA total,s ;If total != 0x8000 then
FE7D A08000 CPWA 0x8000,i
FE80 18FE8A BREQ L6
FE83 08 NEGA ;Negate total
FE84 E3000B STWA total,s
FE87 12FE90 BR setNZ
FE8A C00000 L6: LDWA FALSE,i ;else -32768 is a special case
FE8D E30006 STWA isOvfl,s ;isOvfl <- FALSE
;
FE90 DB000F setNZ: LDBX oldNZVC,s ;Set NZ according to total result:
FE93 880001 ANDX 0x0001,i ;First initialize NZV to 000
FE96 C3000B LDWA total,s ;If total is negative then
FE99 1CFE9F BRGE checkZ
FE9C 980008 ORX 0x0008,i ;set N to 1
FE9F A00000 checkZ: CPWA 0,i ;If total is not zero then
FEA2 1AFE8A BRNE setV
FEA5 980004 ORX 0x0004,i ;set Z to 1
FEA8 C30006 setV: LDWA isOvfl,s ;If not isOvfl then
FEAB 18FEB1 BREQ storeF1
FEAE 980002 ORX 0x0002,i ;set V to 1
FEB1 FB000F storeF1: STBX oldNZVC,s ;Store the NZVC flags

```

```

FEBC  D0000A  deciErr:  LDBA      '\n',i
FEC1  F1FC16          STBA      charOut,d
FEC4  C0FED1          LDWA      deciMsg,i    ;Push address of message onto stack
FEC7  E3FFFE          STWA      -2,s
FECA  580002          SUBSP     2,i
FECD  24FFDE          CALL     prntMsg      ;and print
FED0  00             STOP      ;Fatal error: program terminates

;
FED1  455252  deciMsg:  .ASCII  "ERROR: Invalid DECI input\x00"
      4F523A
      20496E
      76616C
      696420
      444543
      492069
      6E7075
      7400
    
```

The DECO trap handler

- Outputs the operand of DECO in a format that is equivalent to the C `printf()` function with an integer value
- Outputs at most five characters, preceded by the hyphen character – if necessary

```

;***** Opcode 0x38
;The DECO instruction.
;Output format: If the operand is negative, the algorithm prints
;a single '-' followed by the magnitude. Otherwise it prints the
;magnitude without a leading '+'. It suppresses leading zeros.
;
remain:  .EQUATE 0           ;Remainder of value to output
outYet:  .EQUATE 2           ;Has a character been output yet?
place:   .EQUATE 4           ;Place value for division
;

```

FEED	C000FF	opcode38:LDWA	0x00FF,i	;Assert i, d, n, s, sf, x, sx, sfx
FEED	E1FC11	STWA	addrMask,d	
FEF1	24FC7C	CALL	assertAd	
FEF4	24FCCE	CALL	setAddr	;Set address of trap operand
FEF7	580006	SUBSP	6,i	;Allocate storage for locals
FEFA	C2FC13	LDWA	opAddr,n	;A <- oprnd
FEFD	A00000	CPWA	0,i	;If oprnd is negative then
FF00	1CFF0A	BRGE	printMag	
FF03	D8002D	LDBX	'-',i	;Print leading '-'
FF06	F9FC16	STBX	charOut,d	
FF09	08	NEGA		;Make magnitude positive

FF0A	E30000	printMag:STWA	remain,s	;remain <- abs(oprnd)
FF0D	C00000	LDWA	FALSE,i	;Initialize outYet <- FALSE
FF10	E30002	STWA	outYet,s	
FF13	C02710	LDWA	10000,i	;place <- 10,000
FF16	E30004	STWA	place,s	
FF19	24FF44	CALL	divide	;Write 10,000's place
FF1C	C003E8	LDWA	1000,i	;place <- 1,000
FF1F	E30004	STWA	place,s	
FF22	24FF44	CALL	divide	;Write 1000's place
FF25	C00064	LDWA	100,i	;place <- 100
FF28	E30004	STWA	place,s	
FF2B	24FF44	CALL	divide	;Write 100's place
FF2E	C0000A	LDWA	10,i	;place <- 10
FF31	E30004	STWA	place,s	
FF34	24FF44	CALL	divide	;Write 10's place
FF37	C30000	LDWA	remain,s	;Always write 1's place
FF3A	900030	ORA	0x0030,i	;Convert decimal to ASCII
FF3D	F1FC16	STBA	charOut,d	; and output it
FF40	500006	ADDSP	6,i	;Dallocate storage for locals
FF43	01	RET		

```
;Subroutine to print the most significant decimal digit of the  
;remainder. It assumes that place (place2 here) contains the  
;decimal place value. It updates the remainder.  
;
```

		remain2: .EQUATE 2		;Stack addresses while executing a
		outYet2: .EQUATE 4		;subroutine are greater by two because
		place2: .EQUATE 6		;the retAddr is on the stack
		;		
FF44	C30002	divide: LDWA	remain2,s	;A <- remainder
FF47	C80000		LDWX 0,i	;X <- 0
FF4A	730006	divLoop: SUBA	place2,s	;Division by repeated subtraction
FF4D	16FF59		BRLT writeNum	;If remainder is negative then done
FF50	680001		ADDX 1,i	;X <- X + 1
FF53	E30002		STWA remain2,s	;Store the new remainder
FF56	12FF4A		BR divLoop	
		;		
FF59	A80000	writeNum:CPWX	0,i	;If X != 0 then
FF5C	18FF68		BREQ checkOut	
FF5F	C00001		LDWA TRUE,i	;outYet <- TRUE
FF62	E30004		STWA outYet2,s	
FF65	12FF6F		BR printDgt	;and branch to print this digit
FF68	C30004	checkOut:LDWA	outYet2,s	;else if a previous char was output
FF6B	1AFF6F		BRNE printDgt	;then branch to print this zero
FF6E	01		RET	;else return to calling routine
		;		
FF6F	980030	printDgt:ORX	0x0030,i	;Convert decimal to ASCII
FF72	F9FC16		STBX charOut,d	; and output it
FF75	01		RET	;return to calling routine

The HEXO instruction

- Outputs one word as four hex digits from memory


```
;***** Opcode 0x40
;The HEXO instruction.
;Outputs one word as four hex characters from memory.
;
```

FF76	C000FF	opcode40:LDWA	0x00FF,i	;Assert i, d, n, s, sf, x, sx, sfx
FF79	E1FC11	STWA	addrMask,d	
FF7C	24FC7C	CALL	assertAd	
FF7F	24FCCE	CALL	setAddr	;Set address of trap operand
FF82	C2FC13	LDWA	opAddr,n	;A <- oprnd
FF85	E1FC0F	STWA	wordTemp,d	;Save oprnd in wordTemp
FF88	D1FC0F	LDBA	wordTemp,d	;Put high-order byte in low-order A
FF8B	0C	ASRA		;Shift right four bits
FF8C	0C	ASRA		
FF8D	0C	ASRA		
FF8E	0C	ASRA		
FF8F	24FFA9	CALL	hexOut	;Output first hex character
FF92	D1FC0F	LDBA	wordTemp,d	;Put high-order byte in low-order A
FF95	24FFA9	CALL	hexOut	;Output second hex character
FF98	D1FC10	LDBA	byteTemp,d	;Put low-order byte in low order A
FF9B	0C	ASRA		;Shift right four bits
FF9C	0C	ASRA		
FF9D	0C	ASRA		
FF9E	0C	ASRA		
FF9F	24FFA9	CALL	hexOut	;Output third hex character
FFA2	D1FC10	LDBA	byteTemp,d	;Put low-order byte in low order A
FFA5	24FFA9	CALL	hexOut	;Output fourth hex character
FFA8	01	RET		

;Subroutine to output in hex the least significant nybble of the
;accumulator.

;

FFA9	80000F	hexOut:	ANDA	0x000F,i	;Isolate the digit value
FFAC	B00009		CPBA	9,i	;If it is not in 0..9 then
FFAF	14FFBB		BRLE	prepNum	
FFB2	700009		SUBA	9,i	; convert to ASCII letter
FFB5	900040		ORA	0x0040,i	; and prefix ASCII code for letter
FFB8	12FFBE		BR	writeHex	
FFBB	900030	prepNum:	ORA	0x0030,i	;else prefix ASCII code for number
FFBE	F1FC16	writeHex:	STBA	charOut,d	;Output nybble as hex
FFC1	01		RET		

The STRO instruction

- Outputs a null-terminated string from memory

```
;***** Opcode 0x48
;The STRO instruction.
;Outputs a null-terminated string from memory.
;
```

FFC2	C0003E	opcode48:LDWA	0x003E,i	;Assert d, n, s, sf, x
FFC5	E1FC11	STWA	addrMask,d	
FFC8	24FC7C	CALL	assertAd	
FFCB	24FCCE	CALL	setAddr	;Set address of trap operand
FFCE	C1FC13	LDWA	opAddr,d	;Push address of string to print
FFD1	E3FFFE	STWA	-2,s	
FFD4	580002	SUBSP	2,i	
FFD7	24FFDE	CALL	prntMsg	;and print
FFDA	500002	ADDSP	2,i	
FFDD	01	RET		

```
;***** Print subroutine
;Prints a string of ASCII bytes until it encounters a null
;byte (eight zero bits). Assumes one parameter, which
;contains the address of the message.
```

```

;
msgAddr: .EQUATE 2           ;Address of message to print
;

```

FFDE	C80000	prntMsg:	LDWX	0,i	;X <- 0
FFE1	C00000		LDWA	0,i	;A <- 0
FFE4	D70002	prntMore:	LDBA	msgAddr,sfx	;Test next char
FFE7	18FFF3		BREQ	exitPrnt	;If null then exit
FFEA	F1FC16		STBA	charOut,d	;else print
FFED	680001		ADDX	1,i	;X <- X + 1 for next character
FFF0	12FFE4		BR	prntMore	
					;
FFF3	01	exitPrnt:	RET		

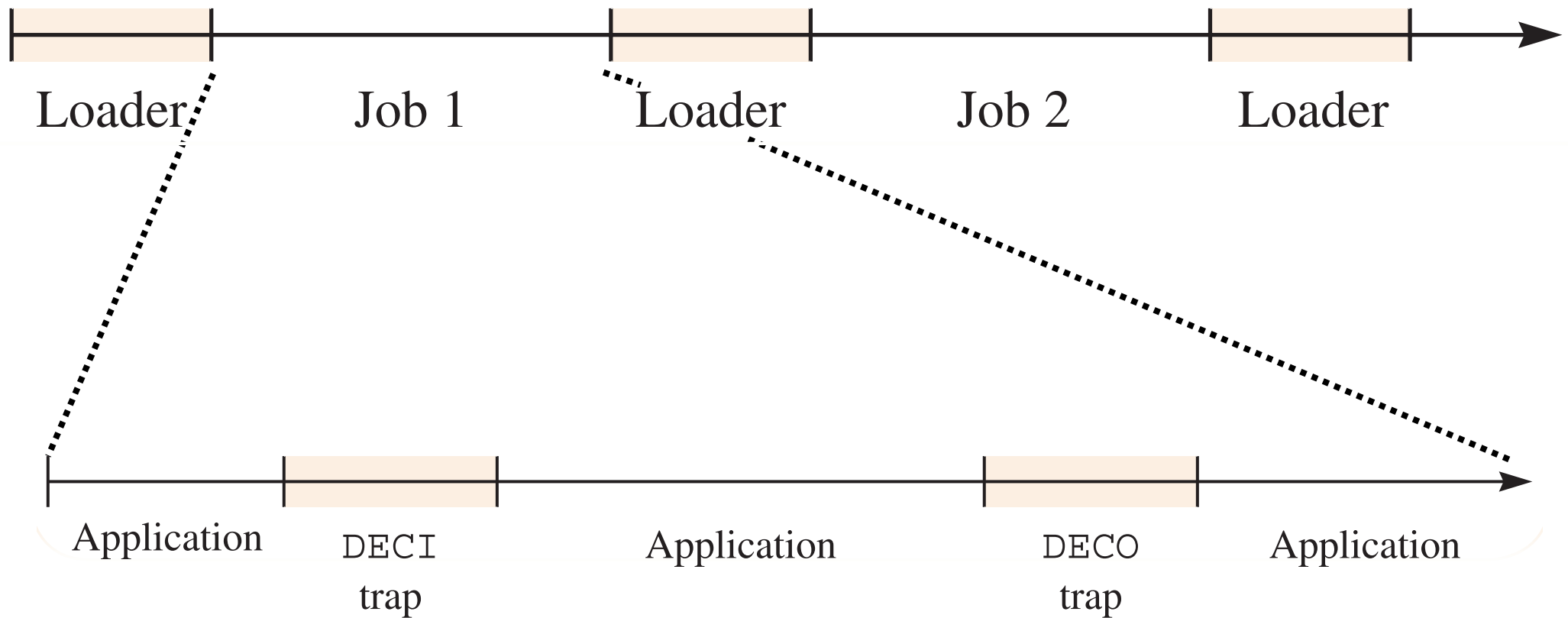
The OS vectors

- Established with `.ADDRSS` command

```

;***** Vectors for system memory map
FFF4  FB8F      .ADDRSS osRAM      ;User stack pointer
FFF6  FC0F      .ADDRSS wordTemp   ;System stack pointer
FFF8  FC15      .ADDRSS charIn     ;Memory-mapped input device
FFFA  FC16      .ADDRSS charOut    ;Memory-mapped output device
FFFC  FC17      .ADDRSS loader     ;Loader program counter
FFFE  FC52      .ADDRSS trap       ;Trap program counter
;
10000          .END
```

OS services

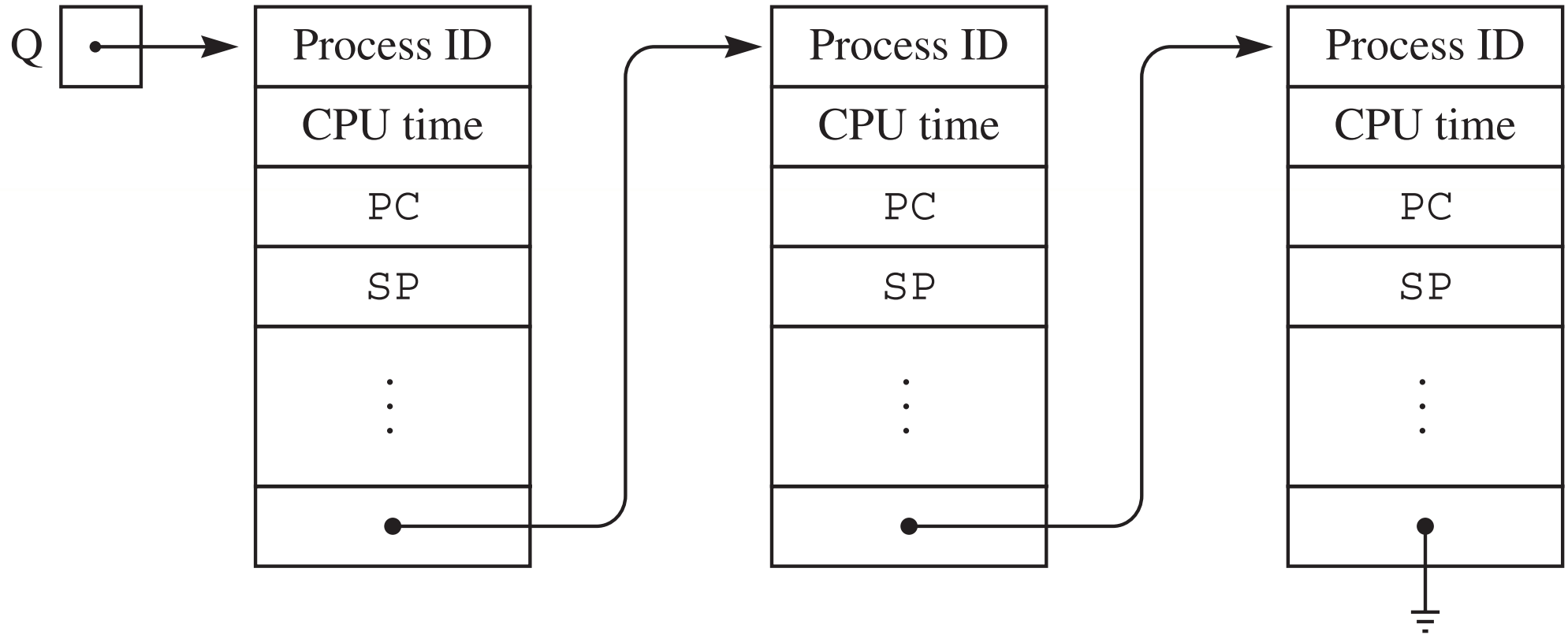


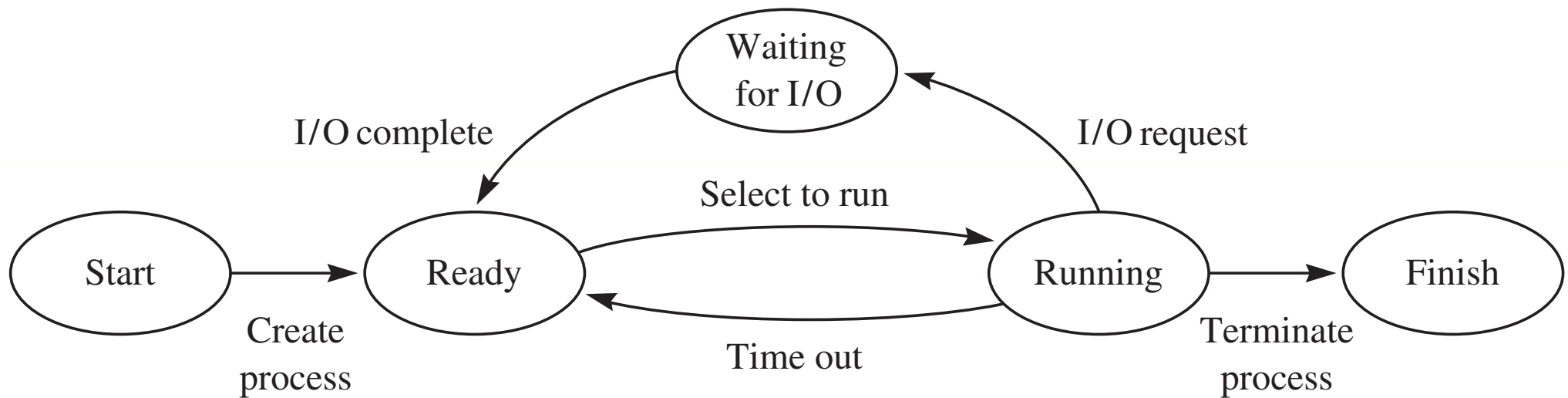
Asynchronous interrupts

- Time outs
- I/O completions

Multiprogramming

- An operating system that can switch back and forth between processes to keep the CPU busy is called a *multiprogramming system*
- It maintains a queue of process control blocks (PCBs)





Multiprocessing

- A computer system with more than one physical CPU
- Also maintains a queue of PCBs, but more than one process can be running at the same time

