Subroutines

- Only one copy of the code is placed in memory
- Whenever we wish to use the code, a jump is made to it
- Jump to address of the first instruction of the subroutine

- Next instruction address should be saved before jump to subroutine is made

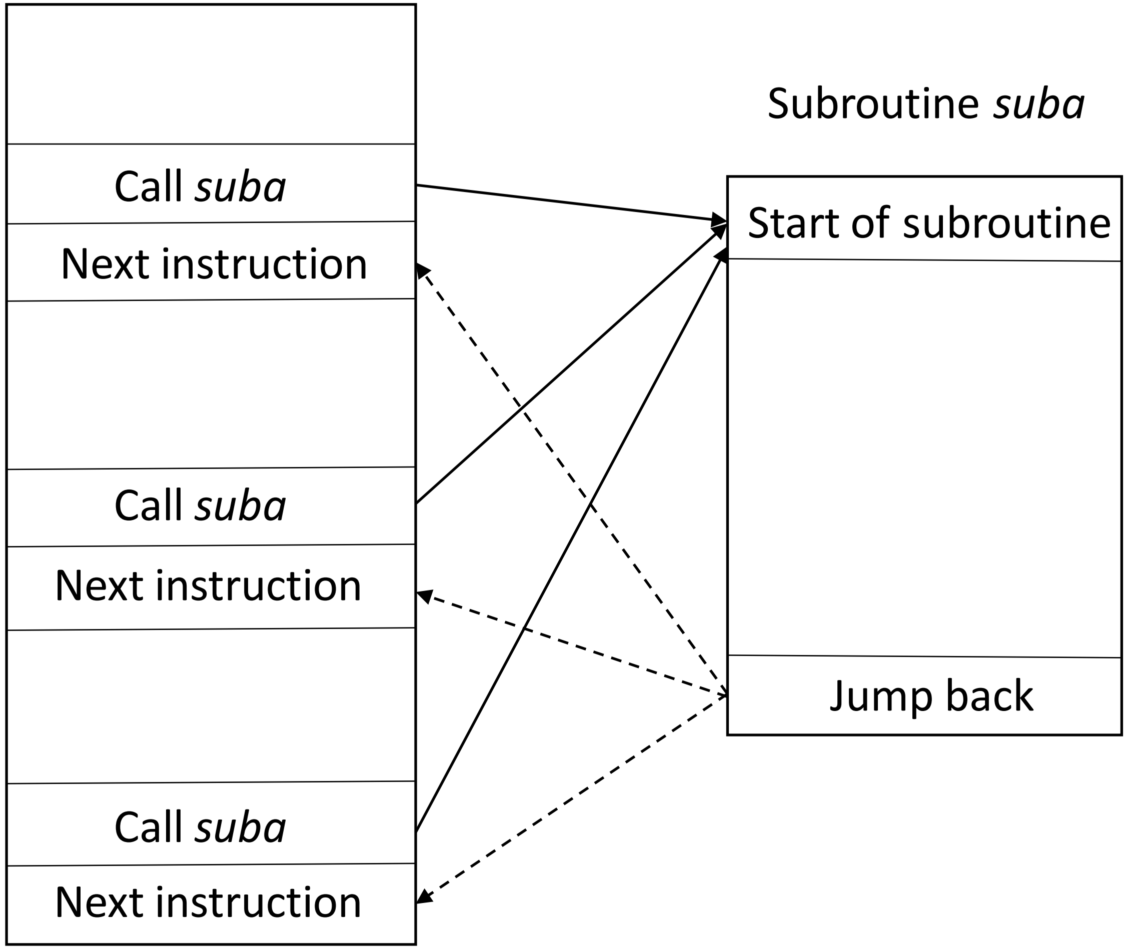


Figure 8.1 of course package

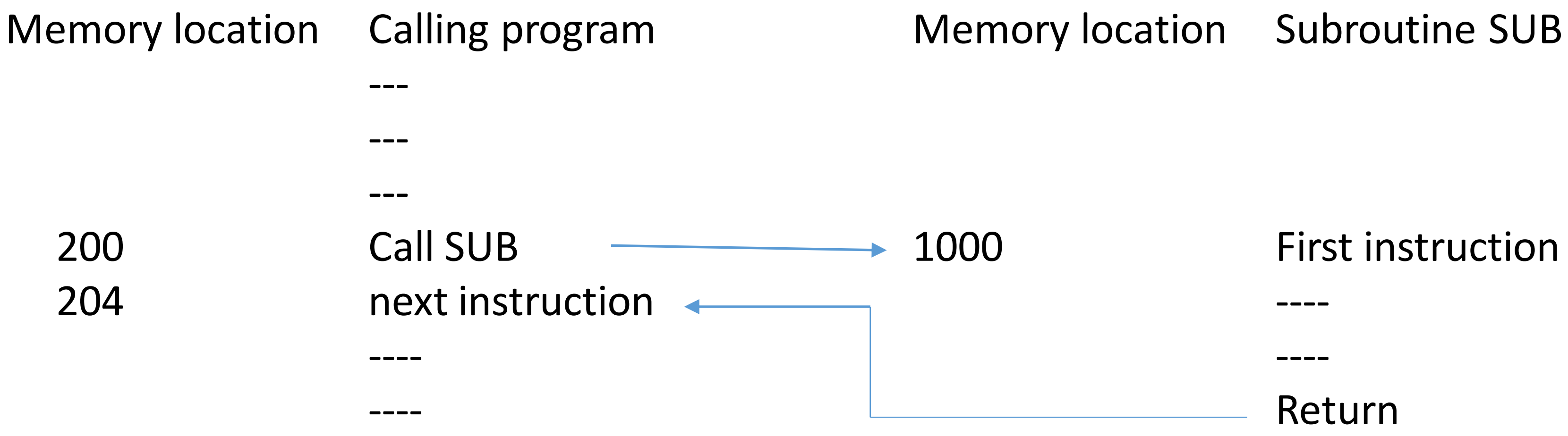
Subroutine calls and returns

```
main        equu        *
;
; first call
;
            move.l      #next1, save_return         ; save return address
            jmp         suba                        ; jump to subroutine
next1       ...         ........                    ; this is where we continue
            ...         ........
;
; second call
;
            move.l      #next2, save_return         ; save return address
            jmp         suba                        ; jump to subroutine
next2       ...         ........                    ; this is where we continue
            ...         ........
;
; third call
;
            move.l      #next3, save_return         ; save return address
            jmp         suba                        ; jump to subroutine
next3       ...         ........                    ; this is where we continue
;
```

```
; Subroutine suba
; suba knows the symbols x and save_return
;

suba        equ         *
            move.w      x,d0
            muls        d0,d0
            move.w      d0,x
            lea         save_return,a0      ;put the correct
                                            ;return address
                                            ;into a0
            jmp         (a0)                ;return

;end of subroutine

save_return             ds.l  4     ;storage for return address
```

Four extra instructions to implement subroutine. Programmer must explicitly save the return address before jumping to subroutine

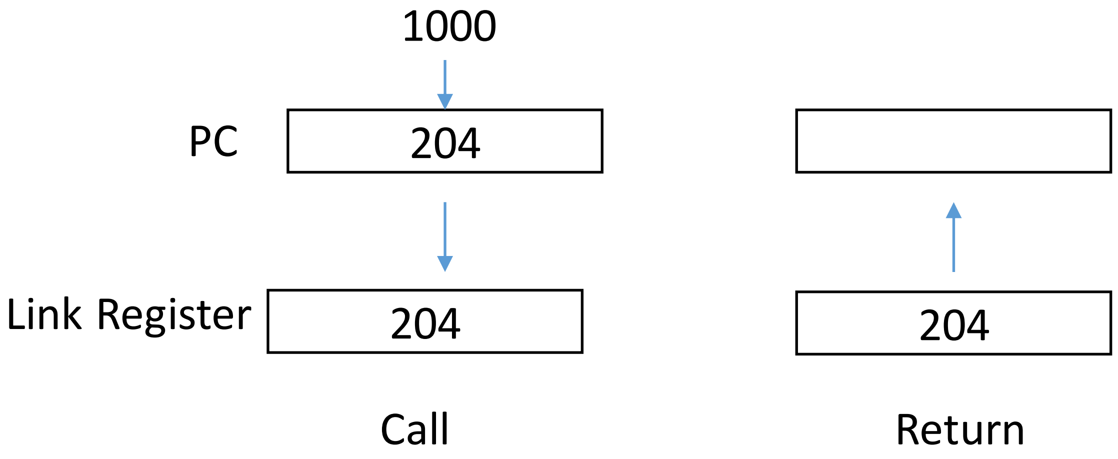
Figure 2.24 [Hamacher] Subroutine linkage using a link register



Memory location	Calling program	Memory location	Subroutine SUB

200	Call SUB	1000	First instruction
204	next instruction		----
	----		----
	----		Return

Here, address of next instruction must be saved by the Call instruction to enable returning to Calling program



Note: Link Register is dedicated to save return address

Nested subroutines

One subroutine calling another

- if link register is used, its previous contents will be destroyed
- it is therefore important to save it in some other location

Stack should be used

- list of similar items arranged in a structure, such that last item added is the first item removed

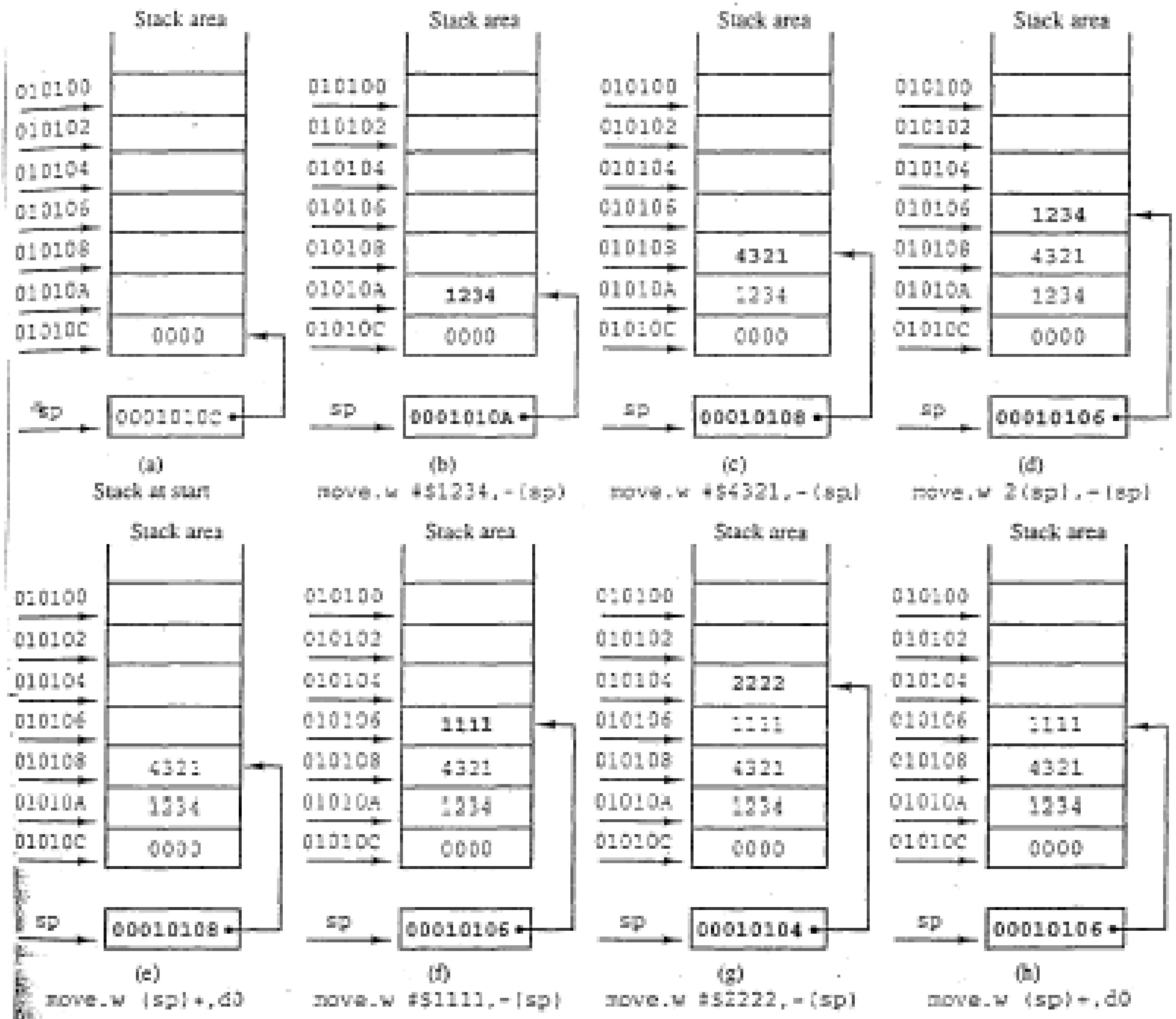
  – Last-in-First-out

- Push an element onto stack
- Pop an element from stack to remove
- elements are either word or longwords

Call instruction – push address of next instruction

Return – pop return address

Stack Pointer originally points to the beginning of the block of memory (Fig 8.2)



(a) Stack at start

(b) move.w #$1234,-(sp)

(c) move.w #$4321,-(sp)

(d) move.w 2(sp),-(sp)

(e) move.w (sp)+,d0

(f) move.w #$1111,-(sp)

(g) move.w #$2222,-(sp)

(h) move.w (sp)+,d0

Figure 8.2 Manipulation of stack using push-and-pop operations.

How to Call Subroutine

Two instructions – jsr, bsr

Jump to subroutine – jsr address (ex. jsr suba)

operand is the Effective Address (specified as absolute address)

- Longword address of the next instruction is pushed on to the stack
- Stack is implicitly used when calling subroutines
- The EA specified is then used to jump to the subroutine

Equivalent machine instruction is (see Fig 8.3):

```
4EB9
0040
0100
```

How to Call Subroutine

Two instructions – jsr, bsr

Branch to subroutine – bsr.b address
bsr.w address (ex. bsr suba)

Same as jsr, except signed displacement is added to PC

Equivalent machine instruction is (see Fig 8.3):

```
(bsr.b)   617E
or
(bsr.w)   6100
          007E
```

Machine instruction contains displacement, calculated using:

Target address = PC + 2 + displacement

Sign displ. added to

Sign displ.

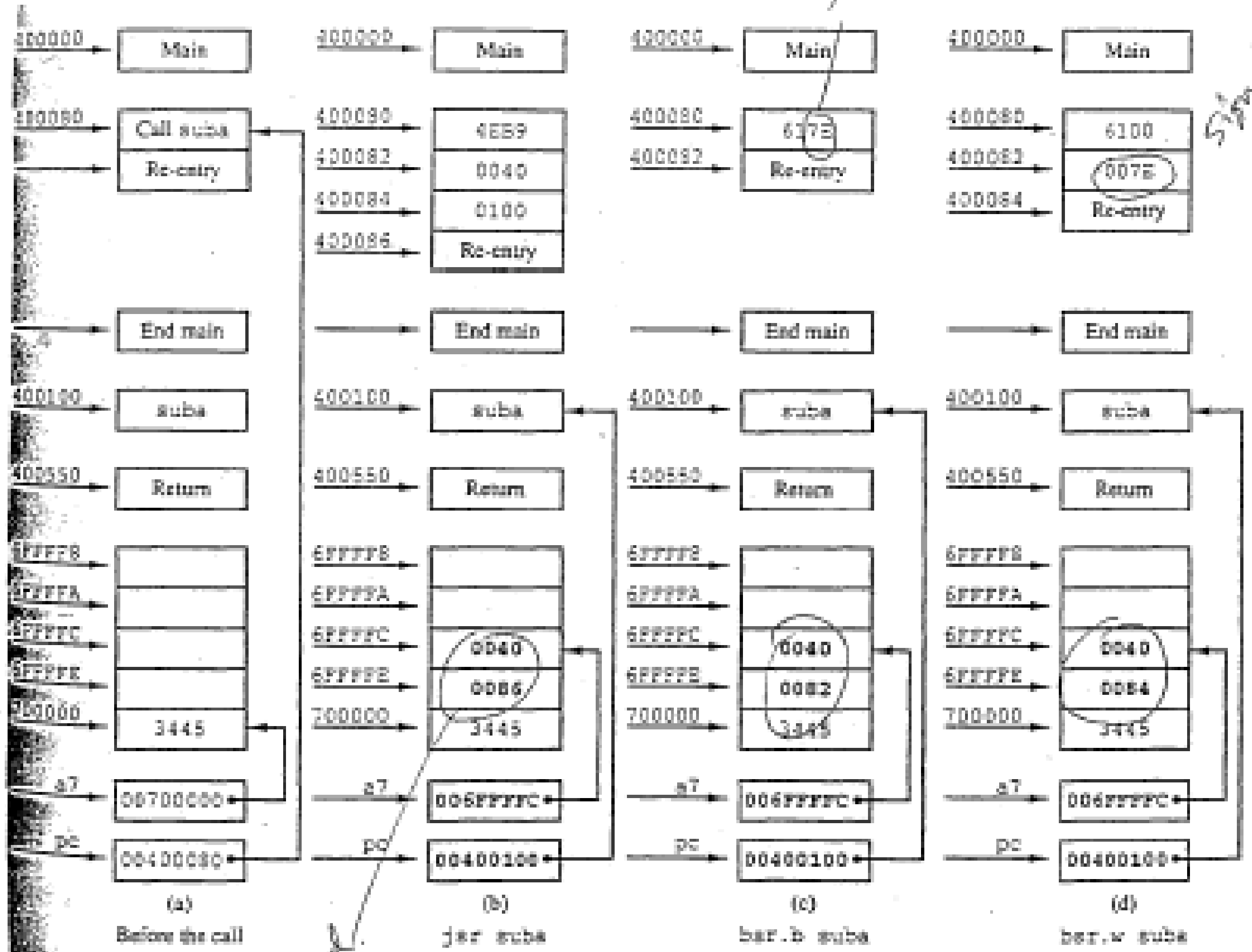


Figure 8.3 Effect of three calls (`jsr`, `bsr.b`, `bsr.w`).

longword address of the next instr is pushed on the stack pointed to by a7

Return from Subroutine

Two ways – rts, rtr

Return from subroutine – rts

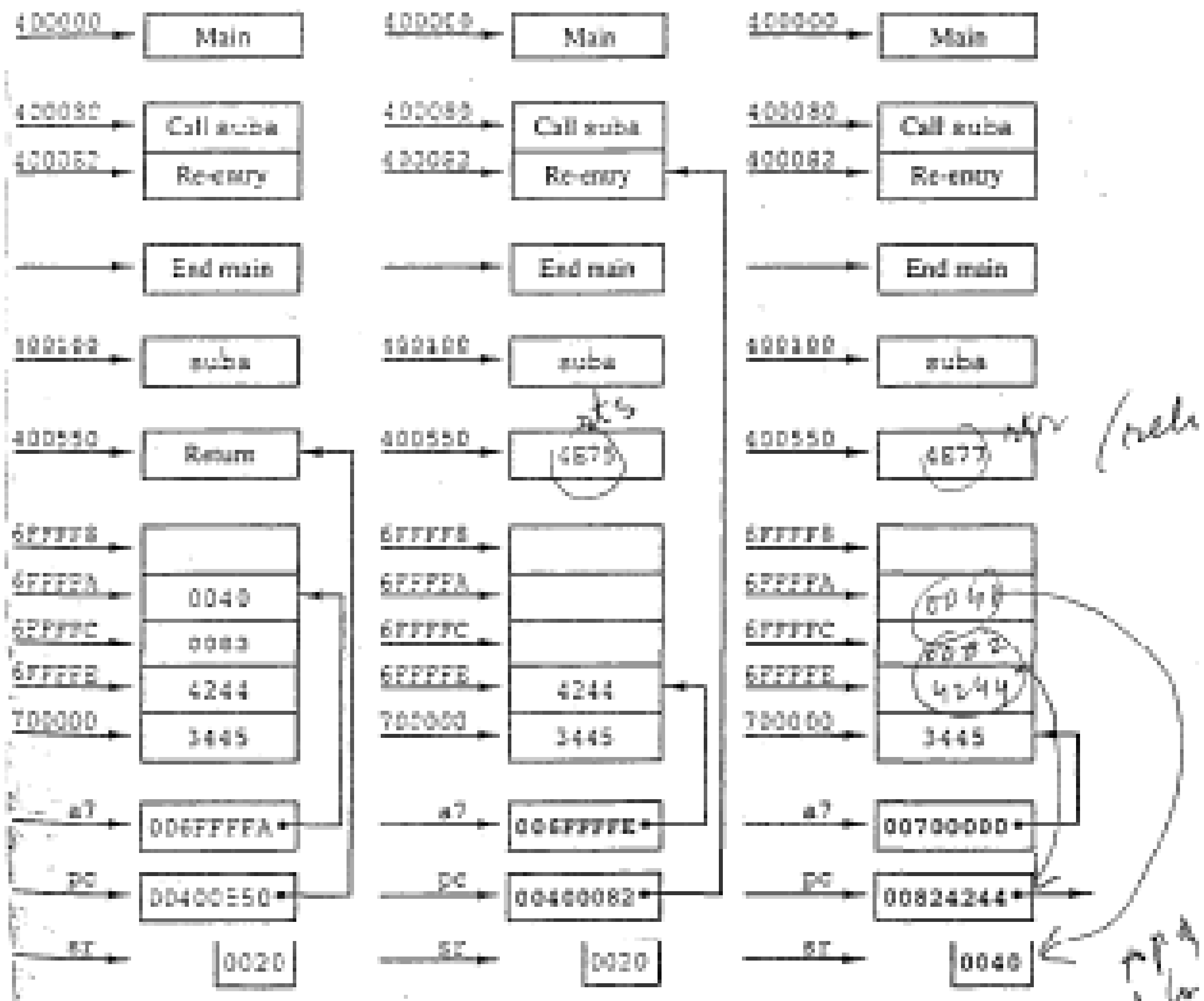
- top of stack is popped off and loaded into PC

Return and Restore – rtr

- first pops a word from stack placing its low byte into CCR (condition code register)
- PC is loaded with next two words popped

If “rtr” is used to return, the subroutine should do the following immediately upon entry to subroutine:

```
move.w  SR, -(SP)
```



(a) Before the return

(b) after rts suba

(c) rtr suba

Ex: Calling and Returning from *suba*

```
main    equ     *
;
; code to make call
;
        jsr     suba    ; first call
next1   ….      …….     ; this is where we continue after return
        ….      …….
        jsr     suba    ; second call
next2   ….      …….     ; this is where we continue after return
        ….      …….
        jsr     suba    ; third call
next3   ….      …….     ; this is where we continue after return
;
```

```
; code of subroutine suba, notice that
; suba knows the symbol x
;
suba    equ     *       ;entry point
        move.w  x,d0
        muls    d0,d0
        move.w  d0, x
        rts
;
; end of subroutine
```