Q1

## Part A

```
   MOV R1, #1     (i = 1)
   MOV R2, #1     (j = 1)
   MOV R3, #32    (k = 32)
   MOV R4, #2

LOOPONE
   CMP R1, #64
   BGE LOOPTWO                  (if i >= 64, exit loop 1)
   STR R4, [R0, R1, LSL, #2]     (array[i] = 2)
   ADD R1, R1, #1                (i = i + 1)
   B LOOPONE                   (repeat loop)

LOOPTWO
   CMP R2, #32
   BGE LOOPTHREE                (if j >= 32, exit loop 2)
   LDR R5, [R0, R2, LSL, #2]    (load array[j] into R5)
   ADD R5, R5, #16             (R5 = array[j] + 16)
   STR R5, [R0, R2, LSL, #2]    (array[j] = array[j] + 16)
   ADD R2, R2, #1              (j = j + 1)
   B LOOPTWO                   (repeat loop)

LOOPTHREE
   CMP R3, #64
   BGE DONE                  (if k >= 64, exit loop 3)
   LDR R5, [R0, R3, LSL, #2]    (load array[k] into R5)
   ADD R5, R5, #32             (R5 = array[k] + 32)
   STR R5, [R0, R3, LSL, #2]    (array[k] = array[k] + 32)
   ADD R3, R3, #1              (k = k + 1)
   B LOOPTHREE                 (repeat loop)

DONE
```

## Part B
LDR time = tpq + tmem + tdec + tmux + tRFread + tALU + tmem + tmux + tRFsetup
= tpq + 2*tmem + tdec + 2*tmux + tRFread + tALU + tRFsetup
= 25 + 2*150 + 70 + 2*25 + 100 + 120 + 46
= 711ps

ADD time = tpq + tmem + tdec + tmux + tRFread + tmux + tALU + tmux + tRFsetup
= 586ps

BL time = tpq + tmem + tdec + tmux + tRFread + tmux + tALU + tmux + tmux
= 565ps

## Part C
exec time = #instructions * CPI * cycle time
CPI = 1 (single-cycle)
cycle time = 711ps (critical path (LDR) time)
#instructions = 4 + 5*63 + 2 + 7*31 + 2 + 7*(64-32) + 2 = 766

therefore exec time = 766 * 1 * 711 = 544626ps = roughly 545ns

Q2

Similar to Exercise 3.30 of textbook with variable names are different.

Solution to Exercise 3.30 of textbook is provided next.

Exercise 3.30

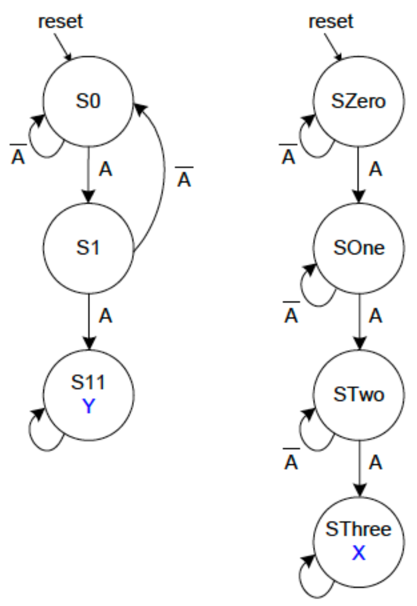


FIGURE 3.11 Factored state transition diagram for Exercise 3.30

current state $s_{1:0}$	input a	next state $s'_{1:0}$
00	0	00
00	1	01
01	0	00

TABLE 3.16 State transition table for output *Y* for Exercise 3.30

current state $s_{1:0}$	input a	next state $s'_{1:0}$
01	1	11
11	X	11

TABLE 3.16 State transition table for output *Y* for Exercise 3.30

current state $t_{1:0}$	input a	next state $t'_{1:0}$
00	0	00
00	1	01
01	0	01
01	1	10
10	0	10
10	1	11
11	X	11

TABLE 3.17 State transition table for output *X* for Exercise 3.30

$$S'_1 = S_0(S_1 + A)$$
$$S'_0 = \overline{S_1}A + S_0(S_1 + A)$$
$$Y = S_1$$

$$T'_1 = T_1 + T_0A$$
$$T'_0 = A(T_1 + \overline{T_0}) + \overline{A}T_0 + T_1T_0$$
$$X = T_1T_0$$

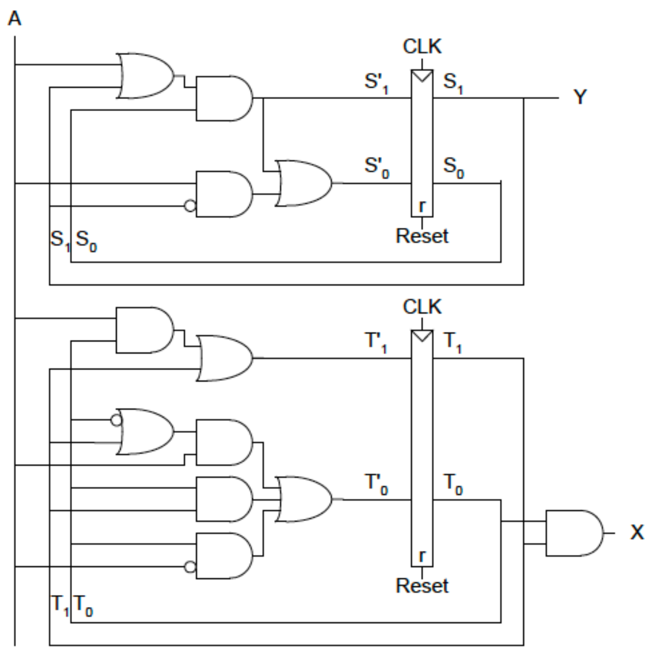


FIGURE 3.12 Finite state machine hardware for Exercise 3.30

Q3

## Part A

1 byte = 8 bits
16 bytes = 128 bits
128/32 = 4 integers per block

## Part B

The I variable and J variable references exhibit temporal locality, as they are referenced repeatedly
in the loops.

The B[I][0] also exhibits temporal locality, as the same array element will be accessed 8000 times as the J loop completes.

The A[I][J] and A[J][I] variables exhibit very (very!) slight temporal locality, as a few of these elements will be accessed multiple times
in the cases when I and J happen to swap.

## Part C
The A[I][J] reference exhibits good spatial locality, as it references each array element in a sequential access pattern.

The B[I][0] reference does not have as good spatial locality, as it jumps between rows.
The A[J][I] does not have good spatial locality, as it moves between rows and columns.

## Part D,E,F (**not relevant, ignore**)

Q4

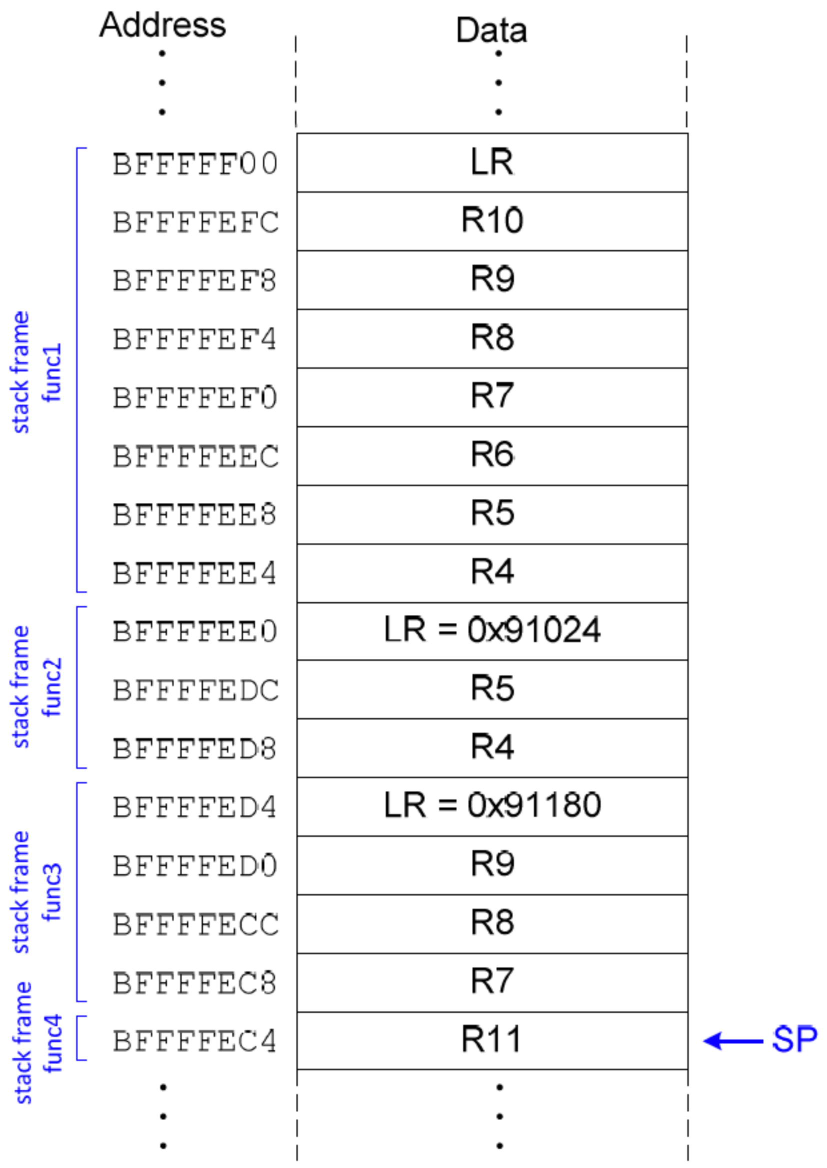
Similar to Exercise 6.27 of textbook. Solution below.

Exercise 6.27

(a)
func1: 8 words (for R4-R10 and LR)
func2: 3 words (for R4-R5 and LR)
func3: 4 words (for R7-R9 and LR)
func4: 1 word (for R11)

(b)

See next page



## Part C (Not relevant)

Q5

## Part A
The QUAC ISA follows these 4 principles extremely well, as it is a simplified ISA designed for teaching purposes (a CISC ISA would be extremely hard to teach)

1) Simplicity favours regularity: the QUAC ISA has 3 instruction encoding formats which are repeated for all instructions.
In addition, the positioning of each register is as similar as possible between the formats (e.g. the destination register is always the first register after the opcode).
This regularity increases the simplicity of the ISA.

2) Smaller is faster: the QUAC ISA only contains a limited number of instructions, which can be used together to complete more complex tasks.

3) good designs demand good compromises: although the ISA would be the most regular if the QUAC only had one encoding format,
this would severely limit the range of instructions that the CPU can complete. For this reason, some regularity is sacrificed in the QUAC ISA as a
compromise to allow a useful range of instructions to be implemented.

4) make the common case fast: the QUAC ISA has a limited number of instructions, and implementing the microarchitecture for these instructions focuses
on making the completion time for these instructions as fast as possible rather than implementing extra, less-used instructions which would slow down the
critical path.

## Part B
Need to add the two's complement of R2 to R1. Need to negate R2, then add 1. Then add this value to R1.

## Part C
A program could conditionally execute based on the carry flag by delegating one of the unused bits in the register operand format to indicate whether to conditionally execute on the carry flag
(wouldn't work for the current immediate format however, as no spare bits). It could then be used similarly to
the zero conditionality, with executing conditionally based on carry shown by the addition of C in the assembly code, e.g.:

MOVL R1, 0xFFFF

```
MOVL R2, Ox1111
MOVL R3, 0x1110

ADD R1, R1, R2     (this instruction will set the carry flag to 1)
SUBC R2, R2, R3    (this instruction will execute, and will set the carry flag to 0)
ANDC R1, R2, R3    (this instruction will not execute, as the carry flag is set to 0)
```

## Part D

```
MOVL R1, #0   (i = 0)
MOVL R2, #8
MOVL R3, DONE (R3 contains current array address)
MOVL R5, #1

LOOP
CMP R1, R2
MOVLZ PC, DONE (if i = 8, move PC to whatever the address of the end of the program is)
LDR R4, [R3]   (load array element)
SUB R4, R4, R1
STR R4, [R3]
ADD R1, R1, 1
ADD R3, R3, 1
MOVL PC, LOOP

ARRAY
.word 1
.word 2
.word 4
.word 10
.word 22
.word 58
.word 233
.word 827

DONE
```