Lab2 Eden Tsubery 316131796 Shai Weinstock 314964727

Q1

1) r[4]=12+45=57
  r[5]=45-r[4]= 45-57=-12
  r[2]=12, r[3]=45 unchanged

2) You can only write to the new register and read from the old register, in the following code there is an attempt to read from a new register and write to an old register.

Q2

1) It will take 6n clock-cycles to execute n instructions because each instruction is executed by going through all states in the state machine.

2) In the proposed solution memory access at every clock cycle is not supported, every reading from memory takes at least 2 clock cycles and they don't happen parallelly.

3) Advantages:
Easy to implement
No hazards that occur from pipelining and dependencies between different instructions
Disadvantages:
At any point of time, only a 1/6 of the cpu is actively processing data
Some instructions dont require all 6 states and still are being executed in 6 clock cycles each.

Q6

We added a struct called dma_t to represent the DMA with the following fields:

- src – source address
- dst – destination address
- len – number of blocks to copy
- state – DMA's FSM current state (IDLE/FETCH/WAIT/COPY)

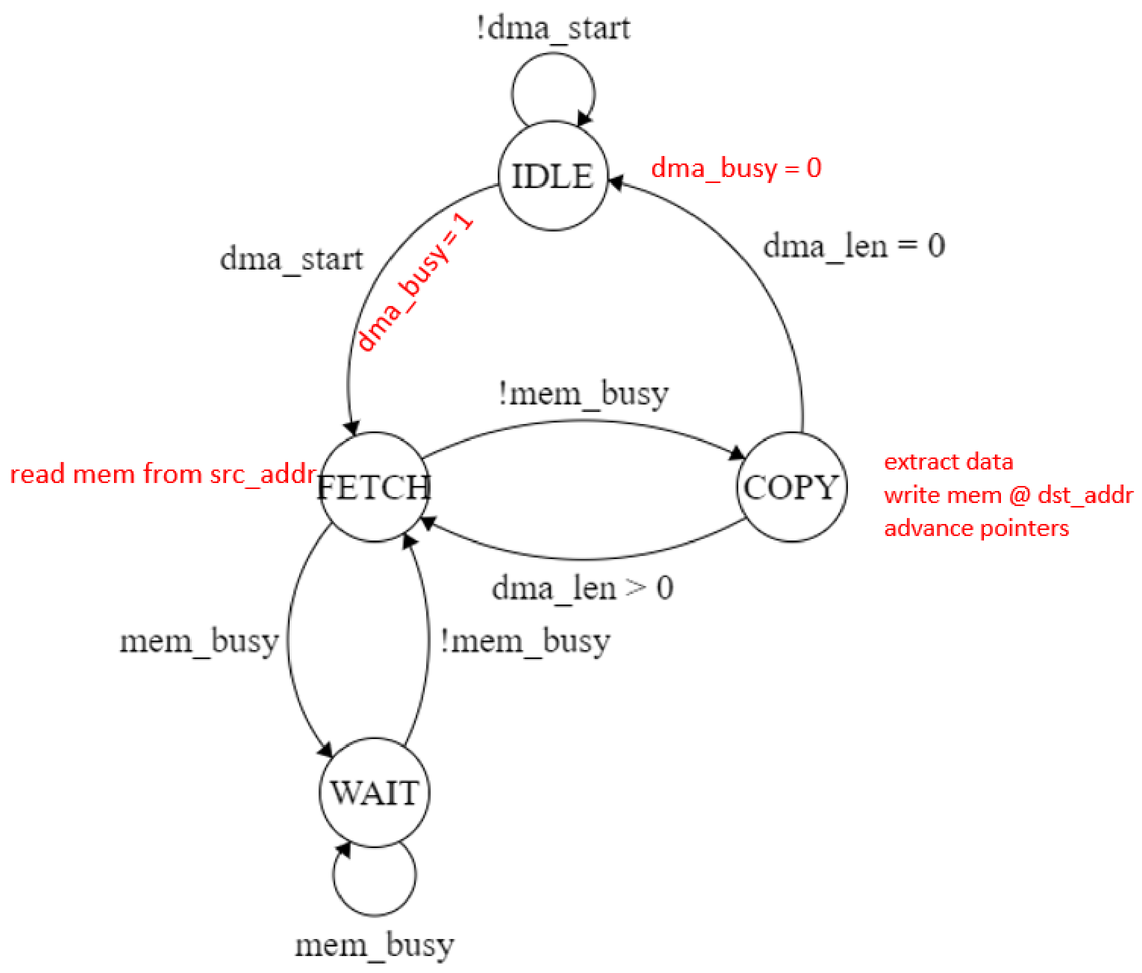
This struct is a field of the sp_s mater structure.
The master structure contains additional control signals:

- dma_start – "kick" to start the DMA operation
- dma_busy – 1 if DMA is in use else 0
- mem_busy – 1 if SRAM is in use (for LD, ST operations) otherwise 0

We have added two new opcodes for the DMA's operation:

- CPY (opcode 10) – used to initiate the DMA transfer
- POL (opcode 11) – used to check transfer status

The DMA works according to the following state machine:



IDLE – When DMA is not in use (set to free)

FETCH – Read data from SRAM (if SRAM is not busy) at current address

WAIT – Hold while SRAM is busy

COPY – Extract read data and write to destination's current address. Update pointers to read and write from the subsequent address, decrement len. When len is zero, free DMA and go back to IDLE state.

The following modifications are done to the SP's states:

- EXEC1 –
  CPY: populate the src, dst and len fields according to the issued command and set the dma_start flag to 1.
  POL: populate R[dst] register with the value of dma_busy.

NOTE: all other SP states (except for IDLE) trigger the dma_ctl() function which calculates the next DMA state based on the current state.

Q7

Test program

```
asm_cmd(ADD, 2,1,0, 1); //set initial result to 1=success
asm_cmd(ST, 0,1,1,200); //set initial memory
asm_cmd(ST, 0,1,1,201);
asm_cmd(ST, 0,1,1,202);
asm_cmd(ST, 0,1,1,203);
asm_cmd(ST, 0,1,1,204);
asm_cmd(ST, 0,1,1,205);
asm_cmd(ST, 0,1,1,206);
asm_cmd(ST, 0,1,1,207);
asm_cmd(ST, 0,1,1,208);
asm_cmd(ST, 0,1,1,209);
asm_cmd(ST, 0,1,1,210);
asm_cmd(ST, 0,1,1,211);
asm_cmd(ST, 0,1,1,212);
asm_cmd(ST, 0,1,1,213);
asm_cmd(ST, 0,1,1,214);
asm_cmd(ADD,3,1,0,210); //R[3]=210
asm_cmd(ADD,0,0,0,0);
asm_cmd(ADD,5,1,0,200);//R[5]=200
asm_cmd(ADD, 6,1,0,15); //R[6]=15
asm_cmd(CPY, 3,5,6,0); //dst=210, src=200, len=15
asm_cmd(POL,4,0,0,0); //while dma_busy
asm_cmd(JNE,0,0,4,21); //()
asm_cmd(LD,5, 0,1,210); //should be overwritten with M[200]=200
asm_cmd(ADD,3,1,0,200);
asm_cmd(JNE, 0,5,3,69);//if not, jump to error
asm_cmd(LD,5, 0,1,211);
asm_cmd(ADD,3,1,0,201);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,212);
asm_cmd(ADD,3,1,0,202);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,213);
asm_cmd(ADD,3,1,0,203);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,214);
asm_cmd(ADD,3,1,0,204);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,215);
asm_cmd(ADD,3,1,0,205);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,216);
asm_cmd(ADD,3,1,0,206);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,217);
asm_cmd(ADD,3,1,0,207);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,218);
asm_cmd(ADD,3,1,0,208);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,219);
asm_cmd(ADD,3,1,0,209);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,220); //should be overwritten with M[210] which was
M[200]=200 because copying is done in ascending order
asm_cmd(ADD,3,1,0,200);
asm_cmd(JNE, 0,5,3,69); //if copy was not done in ascending order jump to
error
```

```
asm_cmd(LD,5, 0,1,221);
asm_cmd(ADD,3,1,0,201);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,222);
asm_cmd(ADD,3,1,0,202);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,223);
asm_cmd(ADD,3,1,0,203);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(LD,5, 0,1,224);
asm_cmd(ADD,3,1,0,204);
asm_cmd(JNE, 0,5,3,69);
asm_cmd(HLT ,0 ,0 ,0 ,0); //finish- R[2]=1
asm_cmd(ADD,2,0,0,0); //error:set result to 0
asm_cmd(HLT ,0 ,0 ,0 ,0); //finish R[2]=0
```