Tu Lam

ECE 375 / Dr. Justin Goins

Dec 2nd, 2020

Homework #4

(Due Date: Dec 4th, 2020)

2. *Consider the implementation of the* `STD Y+q, Rr` *(Store Indirect with Displacement) instruction on the enhanced AVR datapath.*

a) List and explain the sequence of microoperations required to implement `STD Y+q, Rr`. Note that this instruction takes two execute cycles (EX1 and EX2).

Answer: Below is the microoperations required to implement `STD Y+q, Rr`

EX1: $DMAR \leftarrow YH\!:\!YL + q$

EX2: $M[DMAR] \leftarrow Rr$

b) List and explain the control signals and the Register Address Logic (RAL) output for the `STD Y+q, Rr` instruction.

Answer: The following shows the control signals and the RAL output. (On the next page)

EX1: The content from YH:YL are read from the Register File by feeding YH:YL into rA and rB. From there, it feed through input-A into the Address Adder by setting 1 to MG. On the hand, q is feed through input-B into the adder through MF by setting it to 1. Since Adder is adding A + B together, then Adder_f will hold value 00. After the result been add, it feed though the DMAR by setting 1 to MH. Then we can "don't care" about other control signals beside IR_en, PC_en, DM_w, and SP_en as we don't want these data to be overwritten by set it to 0.

EX2: The address in DMAR is routed through MUXE by setting ME to 1 to provide address for the Data Memory. At the same time, the content of Rr is read from the register file by providing Rr to rB and setting MD to 1. Then, Rr is written to the Data Memory by setting DM_w to 1. All other control signals can be "don't cares" except PC_en and SP_en, which need to be set to 0 to prevent PC register and SP register, respectively, from being overwritten.

Control Signals	IF	`STD Y+q, Rr`	
		EX1	EX2
MJ	0	x	x
MK	0	x	x
ML	0	x	x
IR_en	1	0	x
PC_en	1	0	0
PCh_en	0	0	0
PCl_en	0	0	0
NPC_en	1	x	x
SP_en	0	0	0
DEMUX	x	x	x
MA	x	x	x
MB	x	x	x
ALU_f	xxxx	xxxx	xxxx
MC	xx	xx	xx
RF_wA	0	x	0
RF_wB	0	x	0
MD	x	x	1
ME	x	x	1
DM_r	x	x	0
DM_w	0	0	1
MF	x	1	x
MG	x	1	x
Adder_f	xx	00	xx
Inc_Dec	x	x	x
MH	x	1	x
MI	x	x	x

RAL Output	`STD Y+q, Rr`	
	EX1	EX2
wA	x	x
wB	x	x
rA	YH	x
rB	YL	Rr

3. *Consider the implementation of the* `ICALL` *(Indirect Call to Subroutine) instruction on the enhanced AVR datapath.*

a) List and explain the sequence of microoperations required to implement `ICALL`. Note that this instruction takes two execute cycles (EX1 and EX2).

Answer: Below is the microoperations required to implement `ICALL`

EX1: $M[SP] \leftarrow RARL, SP \leftarrow SP - 1$

EX2: $M[SP] \leftarrow RARH, SP \leftarrow SP - 1, PC \leftarrow Z$

b) List and explain the control signals and the Register Address Logic (RAL) output for the `ICALL` instruction. Control signals for the Fetch cycle are given below. Clearly explain your reasoning

Answer: The following shows the control signals and the RAL output.

EX1: The SP will provide the Data Memory by feeding it through ME setting it to 0 and the RARL is then set to 0 in the MI as it is being written in the Data Memory. This will pass RARL into MD by setting it to 0. Then we can "don't care" about other control signals beside IR_en, DW_r, and NPC_en as we do not want these data to be overwritten by set it to 0. Since the SP is being decrement, we would need to set it to 1 into the Inc_dec and set SP_en to 1 to make it latch on. On here, we do not care about PC_en as it will be overwritten in EX2.

EX2: The SP will again provide into the Data Memory by feeding 0 into the ME. This time the RARH is set to 1 into the MI as it traverses through and going to the Data Memory, but MD is still being a 0. Since the SP is being decrement, we would need to set it to 1 into the Inc_dec and set SP_en to 1 to make it latch on. At the same time the rA and rB hold the address of Z and will pass onto the Adder and passing it through MG as input-1, set adder to 11, and pass it to PC through MJ as input-1. Finally, we can "don't care" about other control signals beside PCh/Phl_en, and DM_r as we do not want these data to be overwritten by set it to 0. We can set IR_en to "Don't care" as this is the last instruction.

Control Signals	IF	ICALL	
		EX1	EX2
MJ	0	x	1
MK	0	x	x
ML	0	x	x
IR_en	1	0	x
PC_en	1	x	1
PCh_en	0	0	0
PCl_en	0	0	0
NPC_en	1	0	x
SP_en	0	1	1
DEMUX	x	x	x
MA	x	x	x
MB	x	x	x
ALU_f	xxxx	xxxx	xxxx
MC	xx	xx	xx
RF_wA	0	0	0
RF_wB	0	0	0
MD	x	0	0
ME	x	0	0
DM_r	x	0	0
DM_w	0	1	1
MF	x	x	x
MG	x	x	1
Adder_f	xx	xx	11
Inc_Dec	x	1	1
MH	x	x	x
MI	x	0	1

RAL Output	ICALL	
	EX1	EX2
wA	x	x
wB	x	x
rA	x	ZH
rB	x	ZL

4. *Consider the implementation of the* `LPM r7, Z` *(Indirect Call to Subroutine) instruction on the enhanced AVR datapath.*

a) List and explain the sequence of microoperations required to implement `LPM r7, Z`. Note that this instruction takes two execute cycles (EX1, EX2, and EX3).

Answer: Below is the microoperations required to implement `LPM r7, Z`

EX1: $PMAR \leftarrow ZH{:}ZL$

EX2: $MDR \leftarrow M[PMAR]$

EX3: $R7 \leftarrow MDR$

b) List and explain the control signals and the Register Address Logic (RAL) output for the `LPM r7, z` instruction. Control signals for the Fetch cycle are given below. Clearly explain your reasoning

Answer: The following shows the control signals and the RAL output.

EX1: The content from ZH:ZL are read from the Register File by feeding YH:YL into rA and rB. From there, it feed through by latching on the PMAR. This is done by feeding it through input-A into the Address Adder by setting 1 to MG. On the hand, since we are passing only A through, then Adder_f will hold value 11. Then we can "don't care" about other control signals beside IR_en, PC_en, DM_w, and SP_en as we don't want these data to be overwritten by set it to 0. And we care about RF_wA and RF_wB by set it as 0.

EX2: The program memory is then reading the content in the PMAR by setting 1 to the ML and latch that value into the MDR. Then we can "don't care" about other control signals beside IR_en, PC_en, DM_w, and SP_en as we don't want these data to be overwritten by set it to 0. And we care about RF_wA and RF_wB by set it as 0.

EX3: We will have the content in MDR is written out into R7 by setting MC to 10. Then we write RF_wB as 1 and RF_wA as 0 as data has been set. All the rest of the control signal can be "don't care" except for SP_en, PC_en, and DM_w as 0 so the content in these signals will not be overwritten by other factors. We can also set the IR_en to "don't care" as this is the last cycle in the instruction cycle.

Control Signals	IF	LPM R7, Z		
		EX1	EX2	EX3
MJ	0	x	x	x
MK	0	x	x	x
ML	0	x	1	x
IR_en	1	0	0	x
PC_en	1	0	0	0
PCh_en	0	0	0	0
PCl_en	0	0	0	0
NPC_en	1	x	x	x
SP_en	0	0	0	0
DEMUX	x	x	x	x
MA	x	x	x	x
MB	x	x	x	x
ALU_f	xxxx	xxxx	xxxx	xxxx
MC	xx	xx	xx	xx
RF_wA	0	0	0	0
RF_wB	0	0	0	1
MD	x	x	x	x
ME	x	x	x	x
DM_r	x	x	x	x
DM_w	0	0	0	0
MF	x	x	x	x
MG	x	1	x	x
Adder_f	xx	11	xx	xx
Inc_Dec	x	x	x	x
MH	x	x	x	x
MI	x	x	x	x

RAL Output	LPM R7, Z		
	EX1	EX2	EX3
wA	x	x	x
wB	x	x	R7
rA	ZH	x	x
rB	ZL	x	x