Rhea Mae Edwards
ECE 375

Assignment #4

Stage	*Micro-Operation*
IF	IR ← M[PC], PC ← PC + 1, NPC ← PC + 1, RAR ← PC + 1

1. Consider the implementation of the **MOVW Rd, Rr** (*Copy Register Word*) instruction on the enhanced AVR datapath.

   (a) List and explain the sequence of micro-operations required to implement MOVW Rd, Rr.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the MOVW Rd, Rd.

   Note that this instruction takes one execute cycle (EX). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

   (a) EX1: Rd + 1:Rd ← Rr + 1:Rr
   (b)

EX1	The contents being read from and written to, Rr + 1 and Rr to rA and rB, and Rd + 1 and Rd to wA and wB. MC is set to 01 to allow Rr to traverse MUXC, and MG to 1 to allow pass through the address adder. Adder_f is set to 11 for move, and both RF_wA and RF_wB is set to 1. PC_en, PCl_en, PCh_en, SP_en, and DM_w are set to 0 to prevent overwrites. Everything else should be "don't care".

Control Signals	IF	MOVW
		EX
MJ	0	x
MK	0	x
ML	0	x
IR_en	1	x
PC_en	1	0
PCh_en	0	0
PCl_en	0	0
NPC_en	1	x
SP_en	0	0
DEMUX	x	x
MA	x	x
MB	x	x
ALU_f	xxxx	xxxx
MC	xx	01
RF_wA	0	1
RF_wB	0	1
MD	x	x
ME	x	x
DM_r	x	x
DM_w	0	0
MF	x	x

MG	x	1
Adder_f	xx	11
Inc_Dec	x	x
MH	x	x
MI	x	x

RAL Output	MOVW
	EX
wA	Rd + 1
eB	Rd
rA	Rr + 1
rB	Rr

Rhea Mae Edwards
ECE 375

2. Consider the implementation of the **ST –X, Rr** (*Store Indirect and Pre-Decrement*) instruction on the enhanced AVR datapath.

   (a) List and explain the sequence of micro-operations required to implement ST –X, Rr.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the ST –X, Rr instruction.

   Note that this instruction takes two execute cycles (EX1 and EX2). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

   (a) EX1: DMAR $\leftarrow$ Xh:Xl, Xh:Xl $\leftarrow$ Xh:Xl – 1
   EX2: M[DMAR] $\leftarrow$ Rr
   (b)

EX1	The contents of Xh and Xl are read from the register file by providing Xh and Xl to rA and rB, and also to wA and wB. This gets latched onto the DMAR register, by setting MH to 0. To be written, MG is set to 01, and MC is set to 01. Adder_f is set to 10 for count, and both RF_wA and RF_wB is set to 1. IR_en, DM_w, PC_en, and SP_en are set to 0 to prevent overwrites. Everything else should be "don't care".
EX2	DMAR needs to provide the address for data memory through MUXE, so ME is set to 1. Rr is written to data memory by setting DM_w to 1. PC_en, PCH_en, PC1_en, and SP_en are set to 0 to prevent overwrites. Everything else should be "don't care".

Control Signals	IF	ST –X, Rr	
		EX1	EX2
MJ	0	xx	xx
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	01	xx
RF_wA	0	1	0
RF_wB	0	1	0
MD	x	x	1
ME	x	x	1
DM_r	x	x	0
DM_w	0	0	1
MF	x	x	x
MG	x	01	xx
Adder_f	xx	10	xx

Inc_Dec	x	x	x
MH	x	0	x
MI	x	x	x

RAL Output	ST –X, Rr	
	EX1	EX2
wA	Xh	x
eB	Xl	x
rA	Xh	x
rB	Xl	Rd

Rhea Mae Edwards
ECE 375

3. Consider the implementation of the **ICALL** (*Indirect Call to Subroutine*) instruction on the enhanced AVR datapath. ICALL is similar to the RCALL (Relative Call to Subroutine) instruction, except that the Z register points to the target address.

   (a) List and explain the sequence of micro-operations required to implement ICALL.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the ICALL instruction.

   Note that this instruction takes two execute cycles (EX1 and EX2). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

   (a) EX1: M[SP] $\leftarrow$ RARl, SP $\leftarrow$ SP – 1
   EX2: M[SP] $\leftarrow$ RARh, SP $\leftarrow$ SP – 1, PC $\leftarrow$ Z
   (b)

EX1	SP needs to provide the address for data memory, so ME is set to 0. MI is set to 0 because of RAR1. To be written, MD is set to 0 and EM_w is set to 1. Adder_f is set to 10 for count, and SP_en is set to 1. IR_en and SP_en is set to 0 for overwrites. Everything else is "don't care".
EX2	SP needs to provide the address for data memory, so ME is set to 0. Since RARh is selected instead, MI must be 1 but MD and DM_w remains the same because it still gets written. Again decrement count by setting Adder_f to 10 and SP_en to 1. Zh and Zl are read in form the register file, which is sent to PC by setting MH to 0, MJ set to 11, and PC_en to 1. Everything else should be "don't care."

Control Signals	IF	ICALL	
		EX1	EX2
MJ	00	xx	11
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	0	0
Adder_f	xx	10	10
Inc_Dec	x	x	x
MH	x	x	0
MI	x	0	1

RAL Output	ICALL	
	EX1	EX2
wA	x	x
eB	x	x
rA	x	Zh
rB	x	Zl

4. Consider the implementation of the **LPM** (*Load Program Memory*) instruction on the enhanced AVR datapath.

   (a) List and explain the sequence of micro-operations required to implement LPM.
   (b) List and explain the control signals and the Register Address Logic (RAL) output for the LPM instruction.

   Note that this instruction takes three execute cycles (EX1, EX2, and EX3). Control signals for the Fetch cycle are given below. Clearly explain your reasoning.

   (a) EX1: PMAR ← Zh:Zl
       EX2: M[PMAR]
       EX3: R0 ← MDR
   (b)

EX1	rA gets Zh and RB gets Zl to read from the register file. This gets latched onto the PMAR register, by setting MH to 0. IR_en, DM_w, PC_en, and SP_en is set to 0 to prevent overwrites. Everything else should be "don't care".
EX2	Program memory gets read to PMAR by setting ML to 1, which is then sent to MDR. IR_en, DM_w, PC_en, and SP_en is set to 0 to prevent overwrites. Everything else should be "don't care".
EX3	R0 get MDR by setting MC to 10 and RF_wB to 1. DM_w, PC_en, and SP_en should be set to 0 to prevent overwrites. Everything else should be "don't care".

Control Signals	IF	LPM		
		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	10
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	x	xx	xx

Adder_f	xx	x	xx	xx
Inc_Dec	x	x	x	x
MH	x	0	x	x
MI	x	x	x	x

RAL Output	LPM		
	EX1	EX2	EX3
wA	x	x	x
eB	x	x	R0
rA	Zh	x	x
rB	Zl	x	x