Computer Architecture Lab4

Lab Assignment 4

Jianyu Huang (UT EID:jh57266)

jianyu@cs.utexas.edu

General Idea

Interruption:

The interruption should be handled when convenient. I choose to handle the interruption in the beginning of the next cycle, so I add a new bit [I] to record the interruption, and once the interruption is detected, the [I] bit will be set. After State 18 of the next cycle, if the microsequenceer finds that [I] is set, the next State will be changed to State 49, which is the entry of the interruption handler.

Exception:

The exception should be handled immediately. I add a micro-op "E<-Check[MAR]" or "E<-Check[IR[15:12]]" in some states which access memory or check opcode and some additional states to check [E] bit, so that the program will go to exception handler once the exception is detected.

I add a new "dummy" state to check [E] bit instead of integrating it into microsequencer so that we can get the next state address in the cycle, because I don't want to modify the microsequencer too much so that the critical path will be extended.

State Diagram Modifications

Inturrption/Exception Handler State Diagram and RTI State Diagram are on Page 3.

The changes of the original state diagrams are shown on Page 4.

Data Path Modifications and New Control Signals

The changes of the original data path and new control signals are shown on Page 5.

Microsequncer Modifications

The changes of the original microsequencer are shown on Page 6. I try to make the changes to microsequencer as little as possible and avoid adding new mux on top of the original microsequencer so that the critical path will not be added.

When the [E] bit is set to 1, which means there is an exception detected in the previous state, the next state address will be set to State 49 by the microsequencer, which is the entry of exception handler. When COND2 is set to 1, if [I] is one, which means there is an interruption in the previous instruction, the program will go to State 49, too. I only add another control signal to the same mux and some combinatorial logic to J[4], so the critical path isn't affected too much.

Technical Details

1. When to reset [E]

Reset_E should be in the same state of [E],

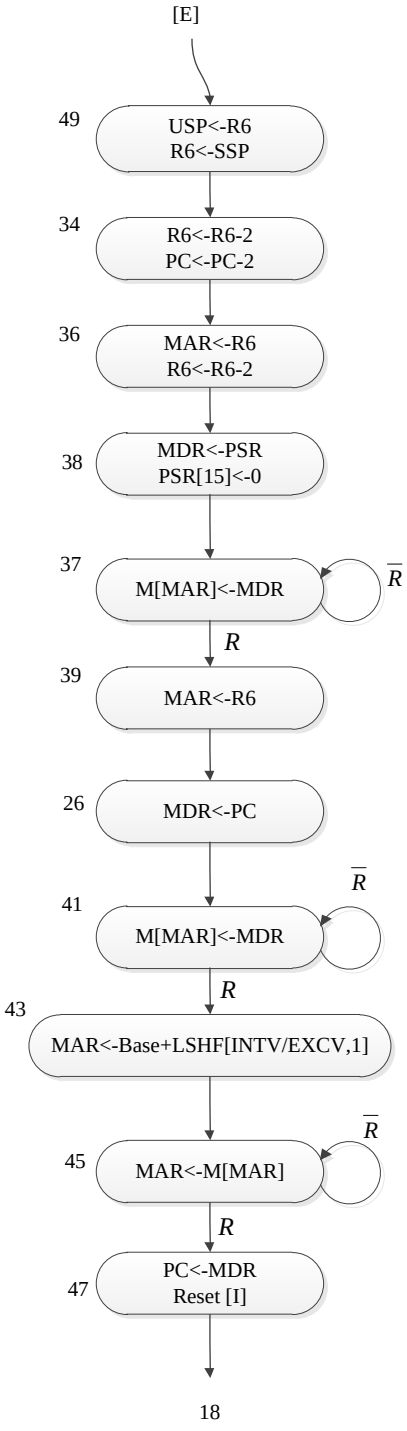
If Reseting E is put on State 49, then there is an issue here: because we first evaluate microsequencer to get the address of the next state before E is reset to 0, we will go to State 49 twice.

2. When to reset [I]

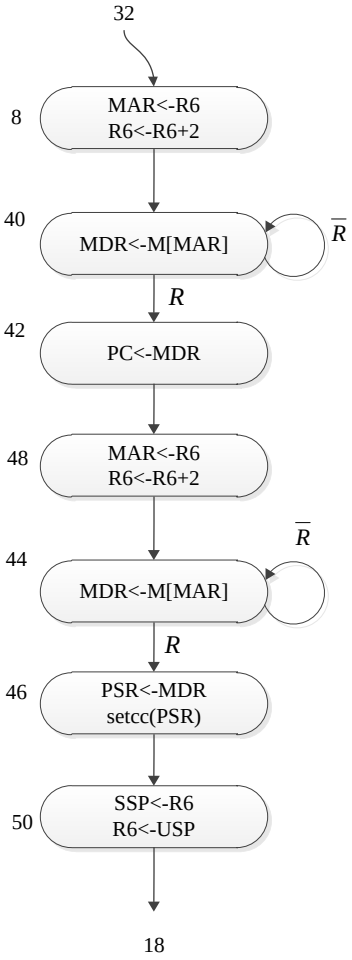
[I] should be reset in the end of interruption handler.

State 43 we need to check whether it is INTV/EXCV. If we have both INTV/EXCV not equal to 0, the only way we can check is through checking [I], so we cannot reset [I] before this state. However, if we can reset EXCV/INTV each time we get out State 43, we may not need to do this way. We only need to check whether EXCV/INTV is 0 or not.

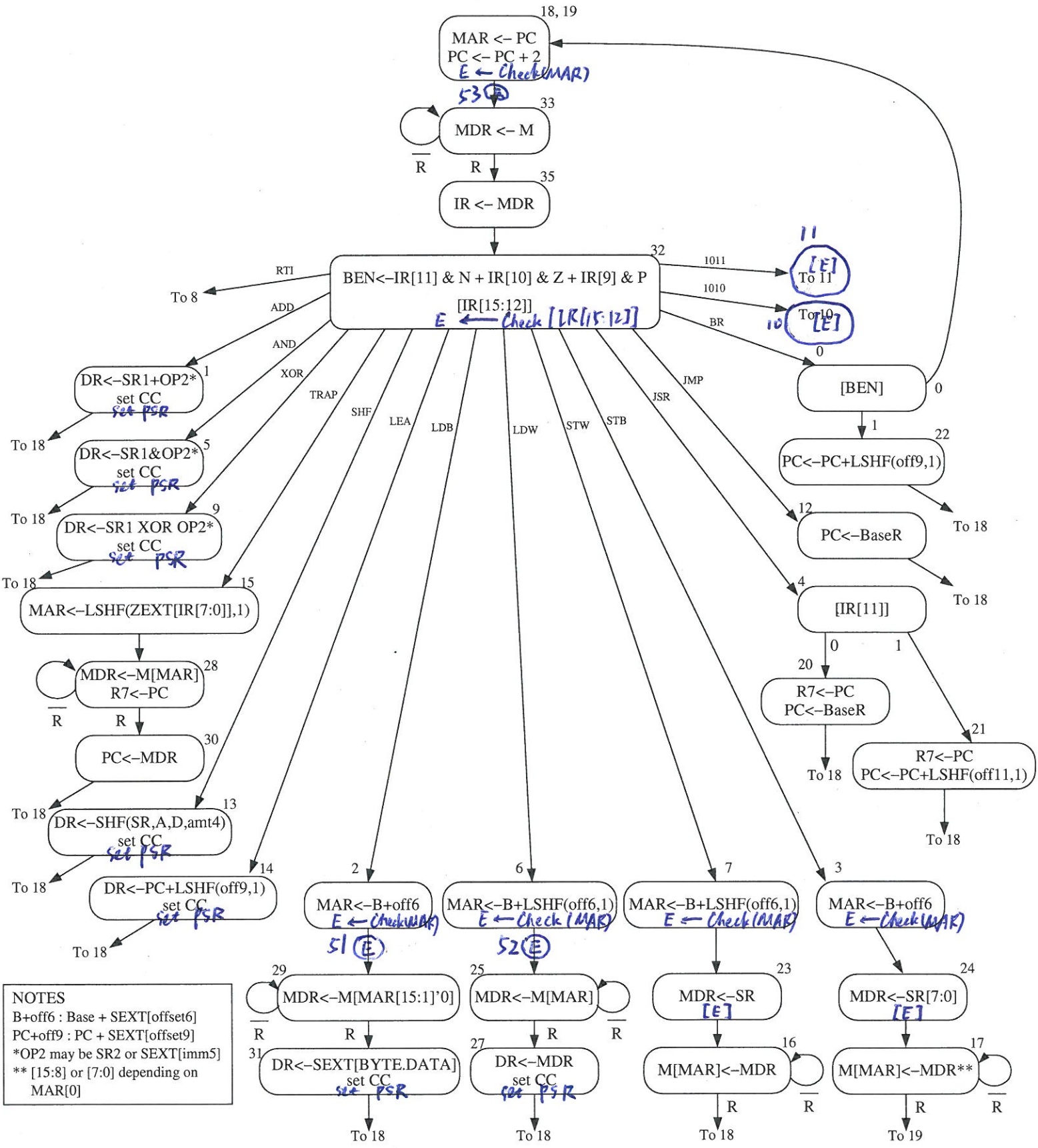
Interruption/Exception Handler



RTI

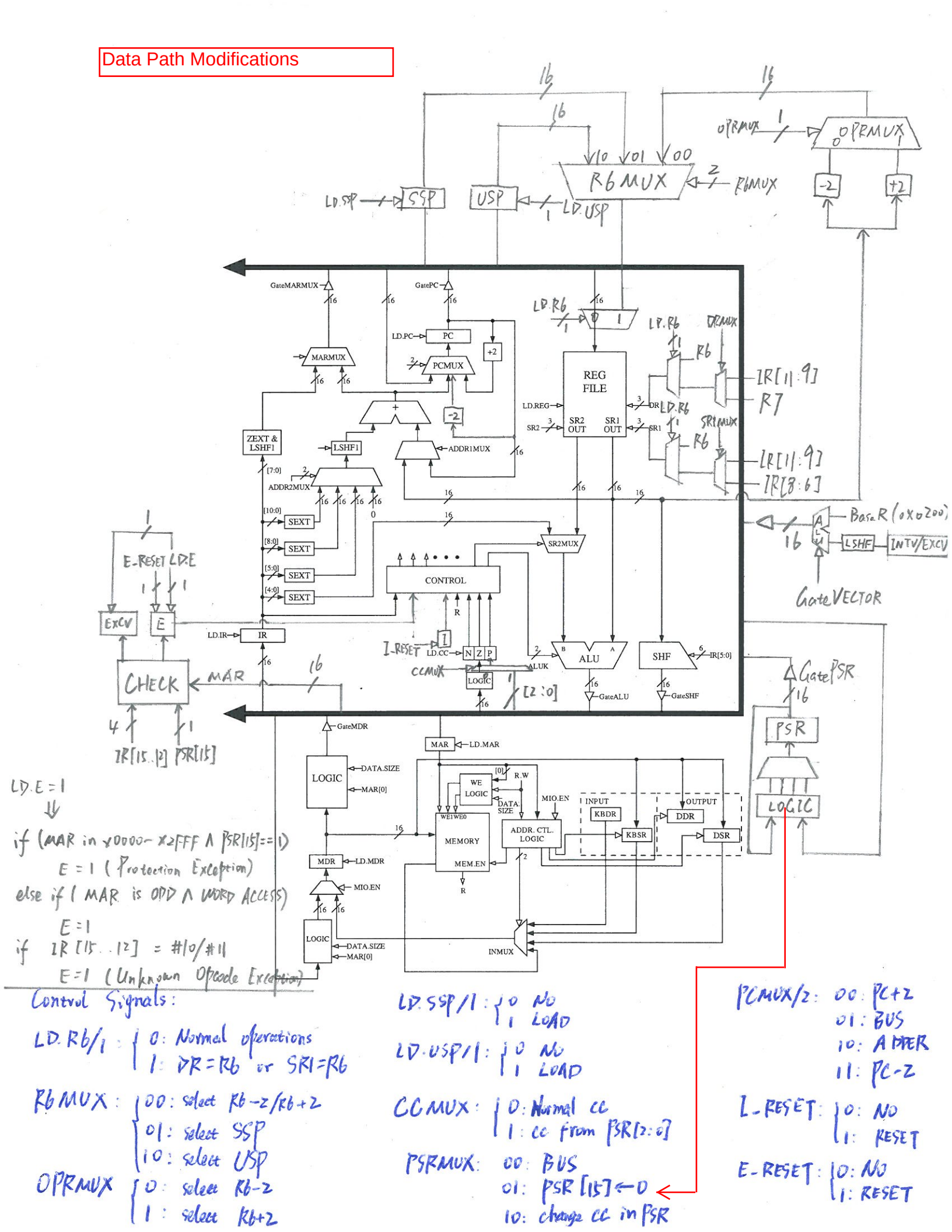


State Diagram Modifications



NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT[offset9]
*OP2 may be SR2 or SEXT[imm5]
** [15:8] or [7:0] depending on MAR[0]

Data Path Modifications



LD.E = 1

⇓

if (MAR in x0000 - x2FFF ∧ PSR[15] == 1)

E = 1 (Protection Exception)

else if (MAR is ODD ∧ WORD ACCESS)

E = 1

if IR[15..12] = #10/#11

E = 1 (Unknown Opcode Exception)

Control Signals:

LD.R6/1 : { 0: Normal operations; 1: DR = R6 or SR1 = R6

R6MUX : { 00: select R6-2/R6+2; 01: select SSP; 10: select USP

OPRMUX { 0: select R6-2; 1: select R6+2

LD.SSP/1 : { 0 NO; 1 LOAD

LD.USP/1 : { 0 NO; 1 LOAD

CCMUX : { 0: Normal cc; 1: cc from PSR[2:0]

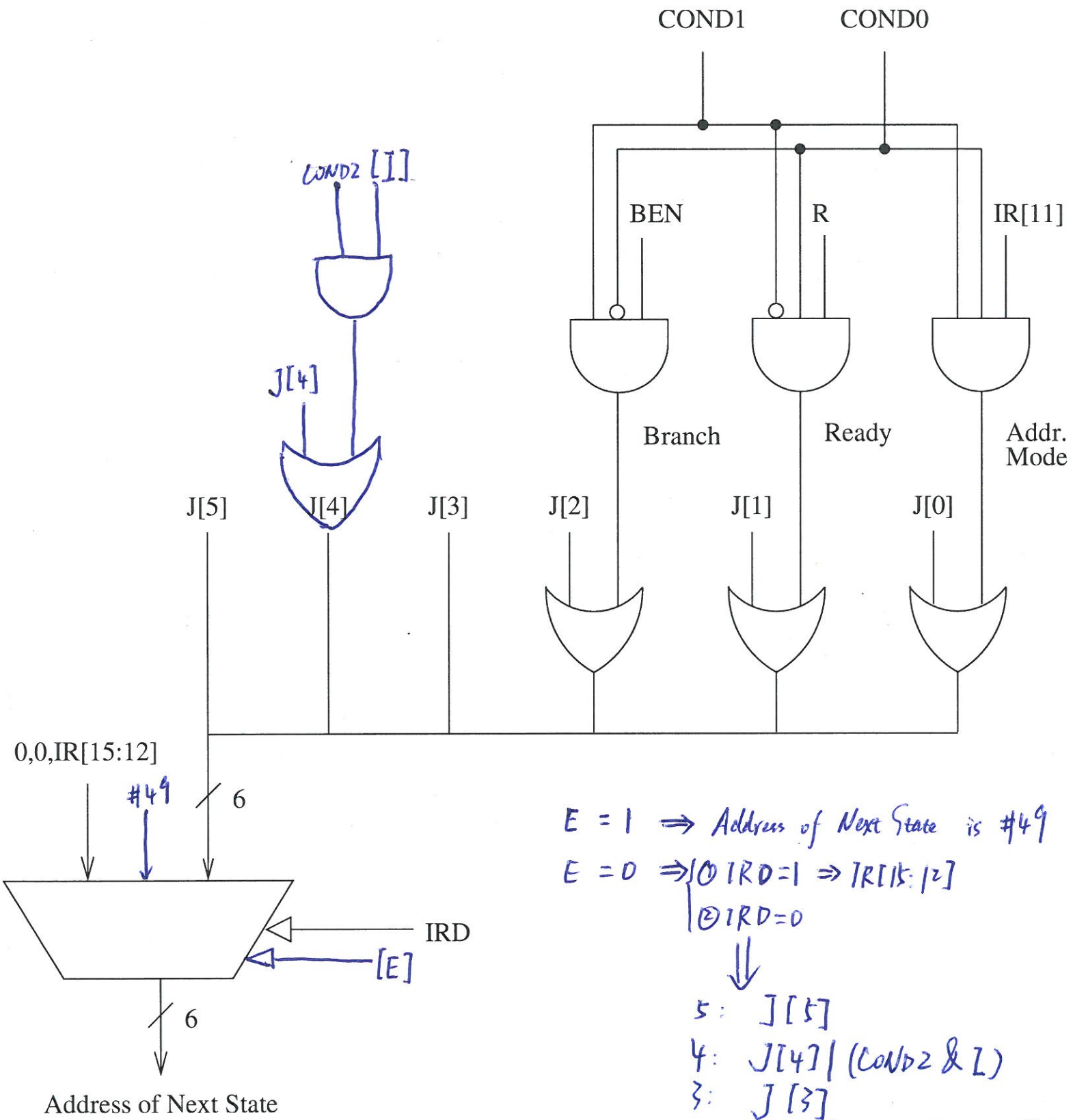
PSRMUX: 00: BUS; 01: PSR[15] ← 0; 10: change cc in PSR

PCMUX/2 : 00: PC+2; 01: BUS; 10: ADDER; 11: PC-2

I_RESET : { 0: NO; 1: RESET

E_RESET : { 0: NO; 1: RESET

Microsequencer Modifications



E = 1 ⇒ Address of Next State is #49

E = 0 ⇒ ① IRD = 1 ⇒ IR[15:12]
② IRD = 0

5: J[5]
4: J[4] | (COND2 & L)
3: J[3]
2: J[2] | (COND1 & $\overline{\text{COND0}}$ & BEN)
1: J[1] | ($\overline{\text{COND1}}$ & COND0 & R)
0: J[0] | (COND1 & COND0 & IR[11])