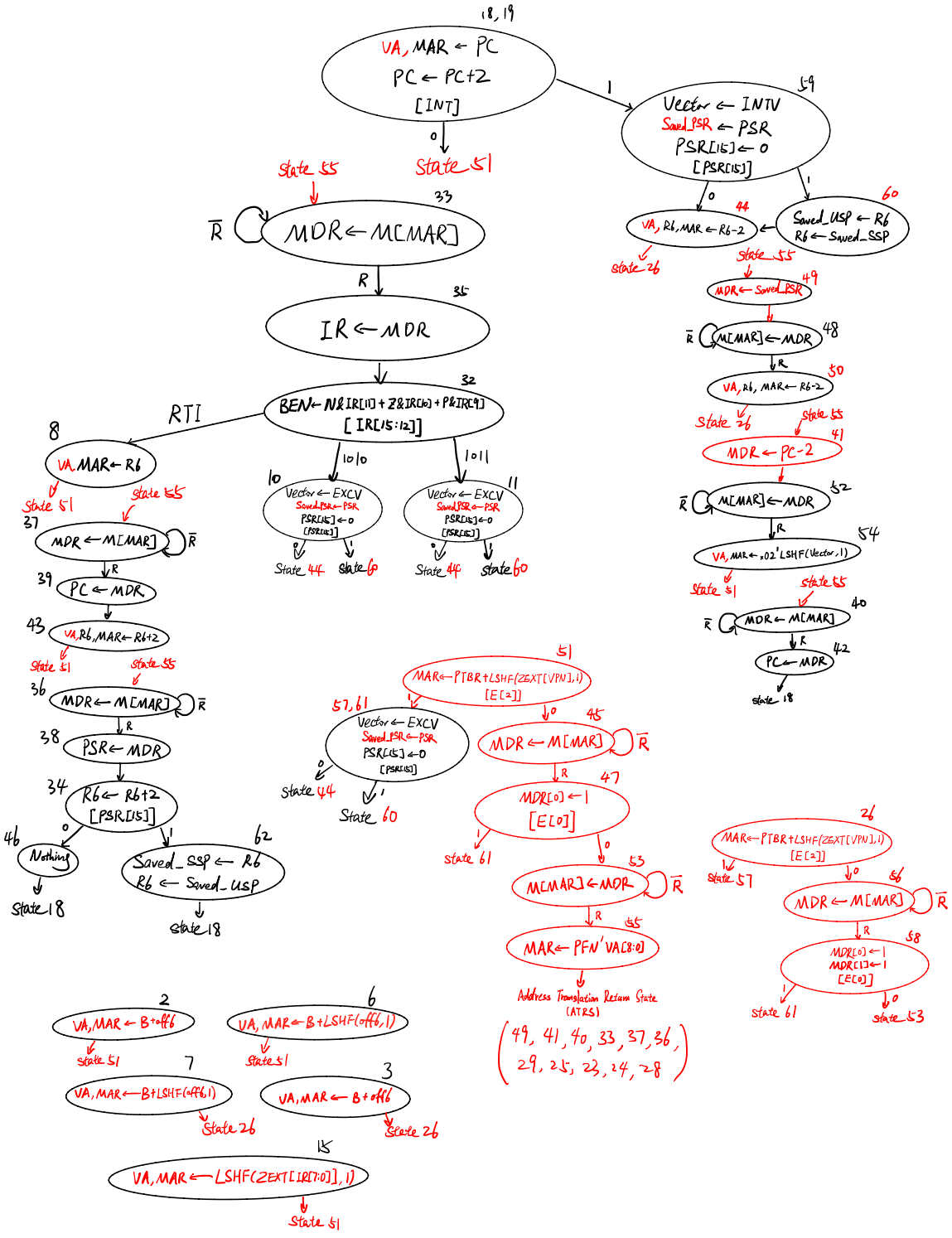
Xinyuan (Allen) Pan
xp572

State Diagram



Xinyuan (Allen) Pan
xp572

The black part is the states added in lab 4 and the red part is either new states or the changes made to the old states.

Overall, changes are made to fit in the address translation process in the state diagram. Address translation is required before every memory access states. Two routes are used for address translation, one is for read accesses (state 51 to 55) and the other is for write accesses (state 26 to 55). They have to be separated because a write access sets the Modify bit in PTE. The address translation states complete the address translation according to the procedure described in the lab documentation, and eventually writes the physical address of the original VA in MAR to MAR, then return to the state where the memory access continues. The return state is stored in the register ATRS (address translation return state) and the mechanism will be explained in Datapath section.

The unaligned access, protection and page fault exceptions are determined during the address translation. Unaligned access exception is determined in state 51 and 26, since the address being checked is VA. Then protection and page fault exceptions are checked in state 47 and 58, where MDR is filled with the PTE. Notice that the Exceptions are determined by a register E, whose logic will be explained later. But what is important here is that E contains the wanted value at the state of 51, 58, 26 and 47.

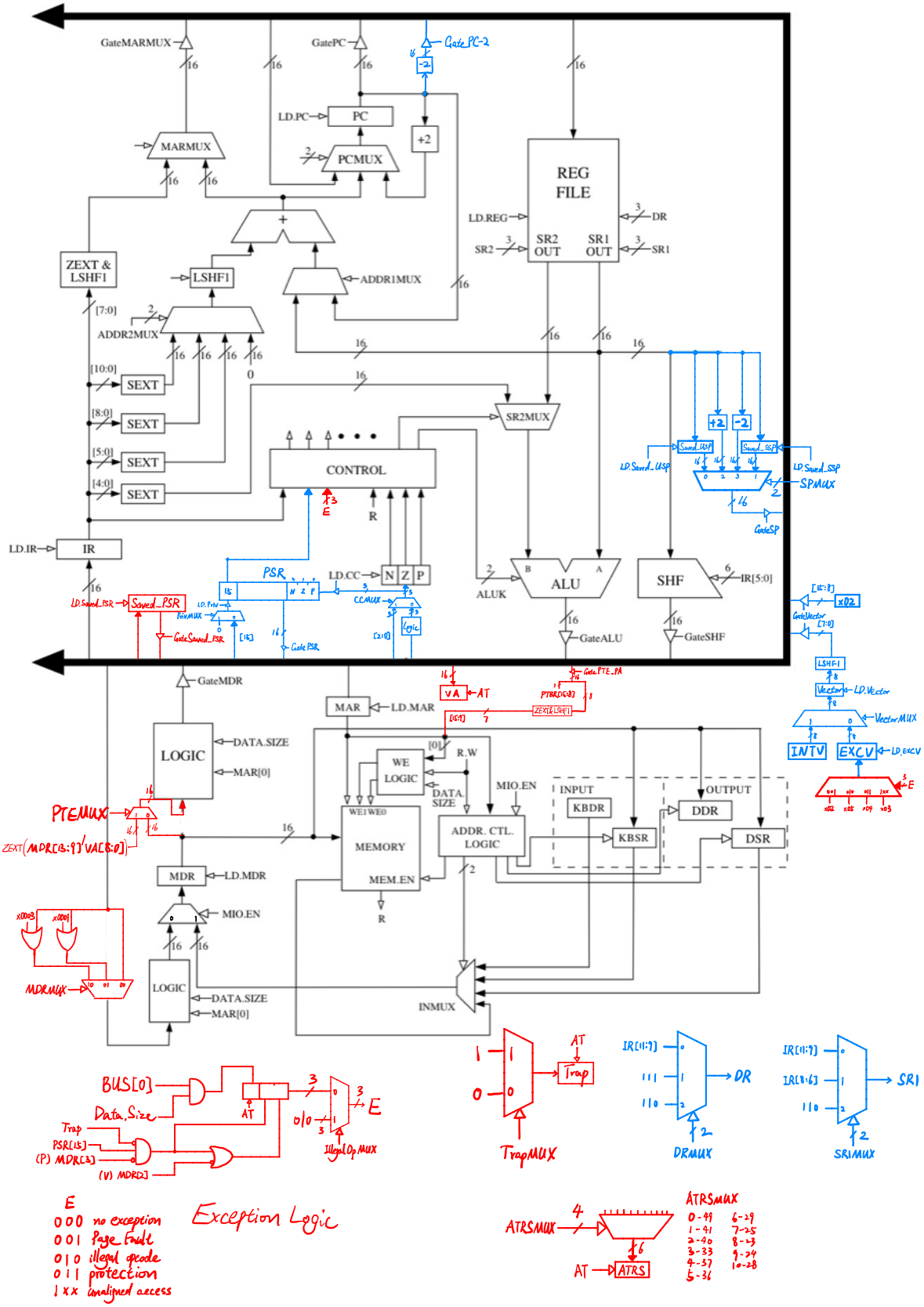
In the states right before entering the address translation, namely states 18,19, 44, 50, 54, 8, 43, 2, 6, 7, 3 and 15, some important preparation is done. They are controlled by 1 control signal, AT (address translation.) Register VA, which holds the virtual address, is set at the same time as MAR, since they have the same value at the time. VA is needed because we will need to retrieve the page offset later in the address translation. Register ATRS is loaded base on a MUX, which selects the return state number. Also, E[2], the bit that determines unaligned access exception, is loaded. Finally, register Trap, which indicates whether the current instruction is a trap instruction, is set based on a control signal TRAPMUX. Trap is needed because the trap instruction can access the protected memory space in user mode.

State 57/61 is the first state in exception handling preparation, and state 59 is the first state in interrupt preparation. They are the same as lab 4, except that now PSR cannot be directly loaded to MDR since MDR will overwritten during address translation. So I added a Saved_PSR register to temporarily store PSR, then Saved_PSR is loaded to MDR in state 59, which is after coming back from the address translation. Another small tweak is swapping the content in state 50 and 41, because again we have to load MDR after the address translation.

The state number encodings are done so that they conform the microsequencer. The details will be discussed in Microsequencer section.

Xinyuan (Allen) Pan
xp572

Datapath & Control Signals



E
000 no exception
001 Page Fault
010 illegal opcode
011 protection
1xx unaligned access

Exception Logic

Xinyuan (Allen) Pan
xp572

Blue part is the components added in lab 4, and red part is either the new components added in lab5 or modified components. I will only be talking about red part in this documentation.

Control Signals:
AT: No, Yes (address translation preparation)
LD.Saved_PSR: No, Yes
GatePTE_PA: No, Yes
GateSaved_PSR: No, Yes
PTEMUX: MDR, ZEXT(MDR[13:9]'VA[8:0])
IllegalOpMUX: Etemp, 010
TrapMux: 0, 1
MDRMUX: from GPR, PTE with set reference bit, PTE with set reference bit & modify bit
ATRSMUX: 49, 41, 40, 33, 37, 36, 29, 25, 23, 24, 28 (address translation return state)
ATR: No, Yes (address translation return, used in microsequencer)

TRAPMUX selects 1 in state 15, and 0 otherwise. And AT signal serves as a load signal for Trap register. The reason that I have to use a Mux to select a value then load that value into Trap register, instead of directly using a control signal to indicate whether it's currently a trap instruction or not, is that the trap instruction shares the address translation states with other instructions, and it is only useful to know whether we are dealing with trap or not during the protection check. We can't have different versions of mircoinstructions for one state, so we have to use a physical component to record whether it's a trap instruction or not. To complete this purpose, Trap register is used in the exception logic.

ATRSMUX selects the state to return to after the address translation, and store it in ATRS register. It's also controlled by AT signal.

Exception Logic:
A 3-bit register E determines the type of exception. E is an input to control unit and will be used in microsequencer.

000 No exception
001 Page fault
010 Illegal opcode
011 Protection
1xx Unaligned access

The exception logic is shown on the lower left corner. E[2] is used for unaligned access, and E[1:0] is used for protection, page fault and illegal opcode. First there is a temporary register. Let's call it Etemp. Notice Etemp[2] is controlled by the AT control signal, so Etemp[2] hence E[2] is only loaded at the certain states. But Etemp[1:0] hence E[1:0] is updated instantly since there are only wires. The reason we need AT to control Etemp[2] is that, UA exception and Protection/PF exception are checked at different time, thus

the value in BUS is not representing the address when we are checking for prot/PF, which in turn may result a 1 in E[2], and E[2] = 1 enforces UA exception. Therefore, E[2] must be guided to load at the proper states. The Trap register is used together with PSR[15], P bit and V bit to determine protection and/or page fault exception. P and V bits are just the bit 3 and 2 in MDR at state 47 and 48. The logic is designed such that when protection and page fault occur simultaneously, E selects protection, 011. E[2] updates in every address translation preparation states, where AT is asserted. Since at those states BUS has the value of MAR, BUS[0] and DATA.SIZE are used to determine UA exception. I update E[2] here instead of state 51 and 26, because we need the information of DATA.SIZE, but in state 51 an 26 the DATA.SIZE is trivial. Finally, an IllegalOpMUX selects between the Etemp and 010, which indicates illegal opcode. One thing worth noting is that IllegalOpMUX and LD.EXCV are set in state 32, rather than state 10 and 11. Because in state 10 and 11 EXCV is loaded into Vector register. I want to ensure that, if two registers, both controlled by LD signals, have data dependency, then they are not loaded in the same cycle. I'm not sure but I think it may affect pipeline process, or it may increase the cycle time. Thus, every time in state 32, E will have the value or 011, but it only has effects if IR[15:12] is 10 or 11.

The E register then serves as the select line for the EXCV mux. It selects the desired vector for the intended exception.

LD.Saved_PSR signal, GateSaved_PSR and Saved_PSR register are explained above in the state diagram section. VA do not need to be gated to the bus because it's placed around the memory. We only pass its value to the PTEMUX. AT serves as the load signal for VA. Also, The value in MAR[15:7] (the VPN) is concatenated with PTBR[15:8] to form the physical address of the PTE, which is gated to the BUS. GatePTE_PA is asserted in state 51 and 26 to load the PTE_PA to MAR.

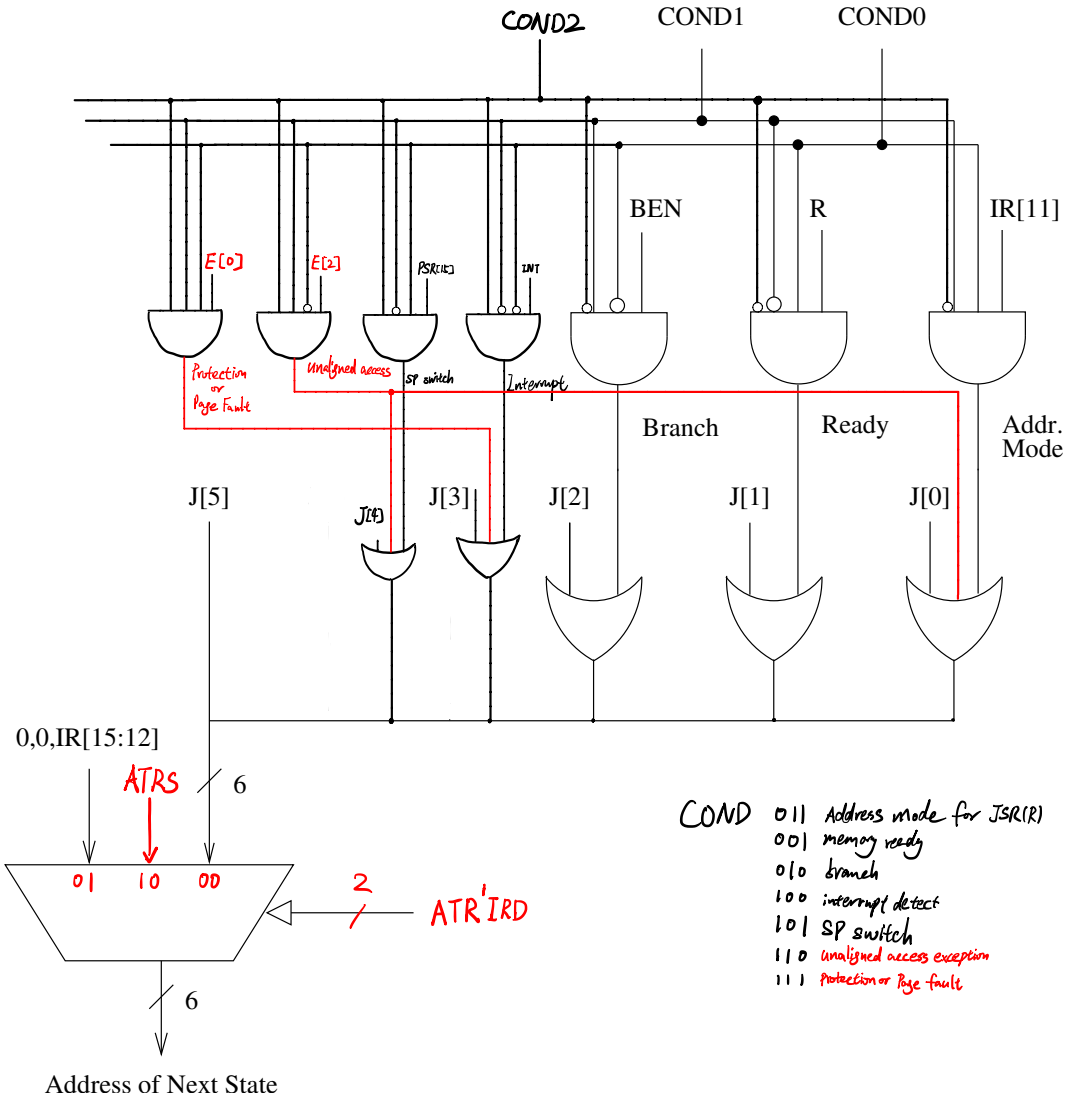
Finally, changes are made to the MDR paths. Before the logic of MDR input, we have the option to modify the R and/or V bits. MDRMUX selects either the original value from bus (which should be a value from general purpose register), or the modified values (which should be the PTE). At the MDR output, MDR is passed to the PTEMUX before entering the output logic. So PTEMUX selects either the original MDR (which should be the regular data requested from memory) or the physical address of the data/code we just translated.

Microsequencer

COND0: unconditional
COND1: memory ready
COND2: branch
COND3: address mode for JSR(R)
COND4: interrupt detect

Xinyuan (Allen) Pan
xp572

COND5: Stack pointer switch
COND6: Unaligned access exception
COND7: Protect or page fault exception
ATR'IRD: 00 uses J and COND
01 state 32, uses 00'IR[15:12]
10 state 55, uses the ATRS



As before, J holds the value for the next state for unconditional states, its own state number for memory access states, and the lower next state number for other states.

ATR and IRD together form a select line to the mux. ATRS contains the state number we are returning to. So when ATR = 1, next state will be ATRS.

COND6 tests E[0] to determine if we have the unaligned access exception, which direct us to the exception preparation state 57/61. The reason we are using 2 states numbers to represent this state is, state number 45 = 101101 in binary and 56 = 111000, and E[2]

is ORed with J[4] and J[0]. So it's going to make 45->111101 = 61, and make 56->111001 = 57. So this way we can avoid expanding control store rows. I ended up having only the state 63 left (BRAVO!)

COND7 tests E[0] to determine if we have a page fault or protection exception, which will direct us to state 61 as well. Since J[3] is ORed with E[0], the (other) next state must be 53.

The state numbers are encoded to conform the state transition mechanism carried out by the microsequencer.