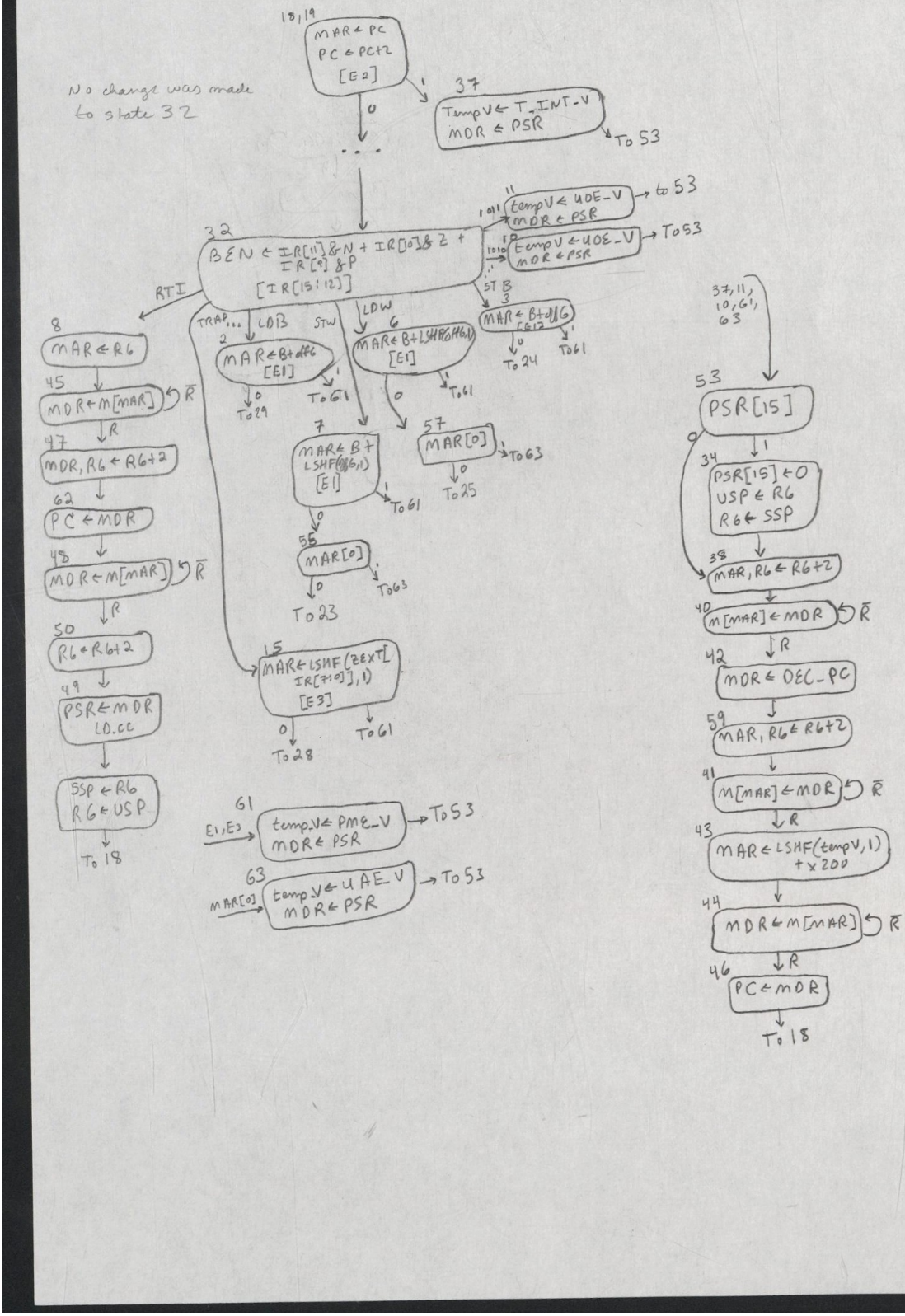
LAB 4 DOCUMENTATION
EE460N

MODIFIED STATE GRAPH
ADDITIONAL DATA PATH STRUCTURES
MODIFIED MICROSEQUENCER

George Neal
gln276

STATE GRAPH


No change was made to state 32
18,19
MAR ← PC
PC ← PC+2
[E2]
37
TempV ← T_INT_V
MDR ← PSR
To 53
32
BEN ← IR[11] &N + IR[10] & Z + IR[9] &P
[IR[15:12]]
RTI
TRAP...
LDB
STW
LDW
STB
1011
tempV ← ILOE_V
MDR ← PSR
to 53
1010
tempV ← UOE_V
MDR ← PSR
To 53
8
MAR ← R6
45
MDR ← M[MAR]
47
MDR, R6 ← R6+2
62
PC ← MDR
48
MDR ← M[MAR]
50
R6 ← R6+2
49
PSR ← MDR
LD.CC
SSP ← R6
R6 ← USP
To 18
2
MAR ← B+dOFF6
[E1]
To 29
To 61
6
MAR ← B+LSHF(OFF6,1)
[E1]
To 61
3
MAR ← B+dOFF6
[E1]
To 24
To 61
7
MAR ← B+LSHF(OFF6,1)
[E1]
To 61
57
MAR[0]
To 63
To 25
55
MAR[0]
To 63
To 23
15
MAR ← LSHF(ZEXT[IR[7:0]],1)
[E3]
To 61
To 28
61
E1,E3
tempV ← PME_V
MDR ← PSR
To 53
63
MAR[0]
tempV ← UAE_V
MDR ← PSR
To 53
37,11,10,61,63
53
PSR[15]
34
PSR[15] ← 0
USP ← R6
R6 ← SSP
35
MAR, R6 ← R6+2
40
M[MAR] ← MDR
42
MDR ← DEC_PC
59
MAR, R6 ← R6+2
41
M[MAR] ← MDR
43
MAR ← LSHF(tempV,1) + x200
44
MDR ← M[MAR]
46
PC ← MDR
To 18
R
W

DESCRIPTION OF NEW STATES IN STATE GRAPH

Included on the state graph are all the new states that enable timer interrupts and the 3 types of exception handling: memory protection, unaligned access, and unknown opcode. State 32 is included for clarity, and remains unchanged from the original LC3b state graph.

Path from state 8 back to 18 (RTI Handling):
This sequence of states restores the machine back to its state before the occurrence of the interrupt/exception that changed the process state from USER to SUPERVISOR. It does so by popping both the old PC and old PSR off of the Supervisor Stack, and latching these values to their respective registers. The sequence also restores the stack pointer (R6) to the User Stack Pointer (value before the interrupt/exception occurred).

NOTE: This implementation makes the following assumptions:

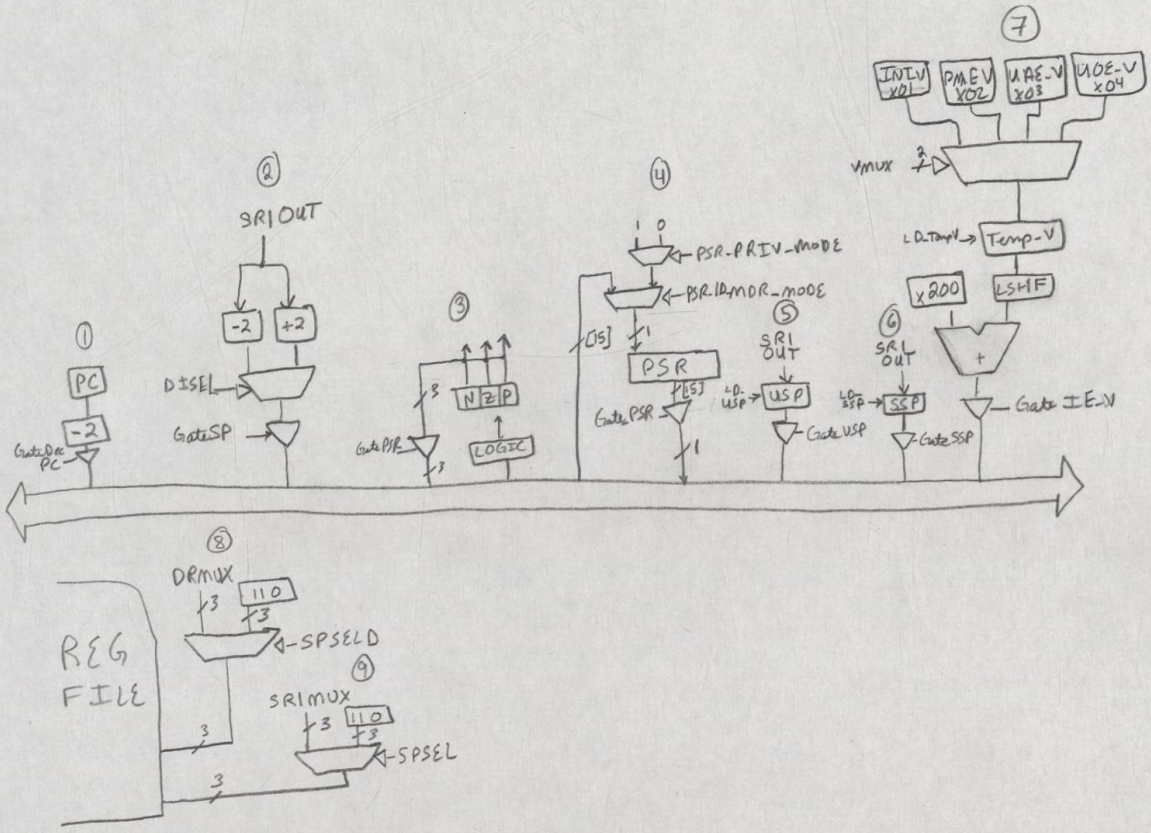
1. RTI is only called when the machine is in SUPERVISOR mode, and was previously running in USER mode.
    1. I.e., nested interrupts and exceptions are not supported.
2. The stack pointer (R6) is pointing to the same location that it was when the machine began handling the interrupt/exception.
3. Interrupt service routines and exception handlers save registers that are clobbered within routines, and restore them at the end of the routines.

Path from state 53 back to 18 (Interrupt/Exception preparation)
This sequence of states switches the state of the machine from USER mode to SUPERVISOR mode (or remain in SUPERVISOR mode if such is the case). The switch from USER to SUPERVISOR mode occurs by switching PSR[15] from a 1 to a 0, storing the Stack pointer to the USP register, and latching the SSP register value to R6. Next, the rest of the current machine state information is saved so that it may be retrieved when the interrupt/exception is serviced. This is done by pushing the decremented PC and PSR onto the Supervisor Stack. Finally, the interrupt/exception vector is left shifted, and added to the base address of the interrupt/exception vector table (x200), and this value is latched to the PC. The machine then returns to state 18, and instruction execution proceeds normally in SUPERVISOR mode.

New interrupt/exception initialization states (10, 11, 37, 61, 63):
These states are microbranched to when the conditions for a timer interrupt or exception are met. Each of them loads the Temp_V register with the appropriate interrupt/exception offset to be later added to the base of the interrupt/exception vector table (x200). Each state also loads the MDR with the current PSR (PSR[15] and Condition Codes).

New Branching states (55, 57):
These states simply check MAR[0] and determine whether or not an unaligned access is occurring. They are new states because the previous states check for protected memory exceptions (protected exceptions have priority to unaligned access exceptions), and must occur in a different state. They could not be placed in the subsequent state either though, because memory is accessed in the subsequent state.

DATA PATH, NEW STRUCTURES

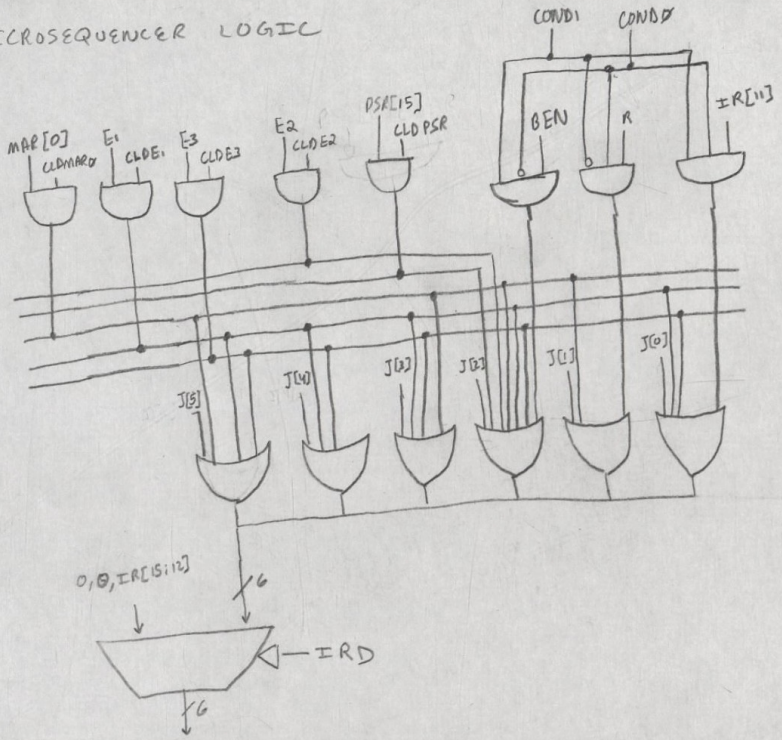


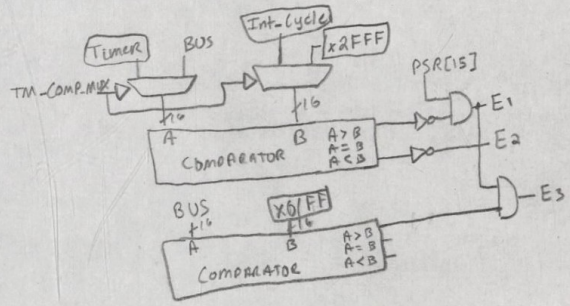
DESCRIPTION OF NEW DATA PATH STRUCTURES

Included in the Data Path sheet are all the structures I added to implement timer interrupts and the 3 exceptions. I will describe the additional structures in order of left to right, top to bottom.

1. The decrementer and gate were added to allow the decremented PC to be pushed onto the Supervisor Stack in state 42.
2. The decrementer and adder are connected to the output of SR1 form the RegFile. The uCode must assert the appropriate signals to make sure the RegFile is outputting R6, the stack pointer. A MUX selects which value to gate to the bus, and the values are used when pushing/popping onto the Supervisor Stack in states 38, 59, 47, 50.
3. The tristate buffer connected to the CC bits was included to save the values when switching control of the machine from USER process to SUPERVISOR process.
4. From top to bottom: a MUX selects which privilege mode to write to the PSR register. Next, another MUX selects either this signal, or BUS[15] (to allow the PSR to be restored from memory in state state 49), and applies either signal to the input of the PSR register. The current PSR register is only 1 bit wide, but may easily be extended to support more features like privilege.
5. USP register saves the User Stack Pointer when switching from USER mode to SUPERVISOR mode.
6. SSP register saves the Supervisor Stack Pointer.
7. From top to bottom. First is a set of registers that include the offsets for the timer interrupt and 3 exceptions for the interrupt/exception vector table. The appropriate offset is chosen based on which event triggered a microbranch to state 10, 11, 37, 61, or 63. The offset is latched into a register, Temp_V in one of these states. Later in the control path (state 43), this value is left shifted and added to the interrupt/exception vector table, and latched to the BUS to be latched into the MDR.
8. This MUX specifies to the RegFile that the destination register is the Stack Pointer (R6).
9. This MUX specifies to the RegFile that the source register (SR1) is the Stack Pointer (R6).

MICROSEQUENCER LOGIC





DESCRIPTION OF NEW STRUCTURES FOR MICROSEQUENCER

Included in the microsequencer logic sheet is the new logic used to implement timer interrupts and the 3 exceptions. The new control bits E1, E2, E3, and MAR[0] and PSR[15] are used to take microbranches in the new states, and are included in the microsequencer.

E1:
The E1 bit signals that a protection exception has occurred. When CLD_E1 is asserted and E1 is high, Jbits 5, 4, 3, 2, and 0 will be set to high, and will branch the current microinstruction to state 61 (assuming J[1] == 0), which prepares the machine to take the protection exception.
E1 is calculated by selecting the values of the BUS and a register containing x2FFF (Bytes[0:0x2FFF] are considered protected) to be compared. If the BUS (driven by what is written to MAR in current cycle) value is less than or equal to x2FFF (not(BUS>x2FFF)) and the current process is in USER mode (PSR[15] == 1), then E1 will be a 1. Otherwise, either (MAR address >= x2FFF) or the process is in SUPERVISOR mode.

E2:
The E2 bit signals that the timer interrupt conditions have been met. When CLD_E2 is asserted and E2 is high, Jbit 4 is set to high, and will branch the current microinstruction (state 33) to state 37, which prepares the machine to take the timer interrupt service routine.
E2 is calculated by comparing the current Timer value to the Int_Cycle value (a constant which indicates which cycle to interrupt on). If the Timer>=Int_Cycle (not(Timer<Int_Cycle)), then E2 is set to high.

E3:
The E3 bit signals that a protection exception has occurred, but is not set to high when the memory location in question is less than x1FF. This is the memory space for TRAP vectors, and is accessible to USER mode processes. When CLD_E3 is asserted and E3 is high, Jbits 5, 4, 3, 2, and 0 will be set to high (assuming J[1] == 0) as the case with E1 above.
E3 is calculated by comparing the value on the BUS (driven by what is being written to MAR) to the value x1FF. If BUS>x1FF, and E1 is currently being driven to high (BUS<=x2FFF and process is in USER mode), then E3 will be set to high.

MAR[0]:
MAR[0] is used to determine whether or not an unaligned access exception should occur. If, when the MAR is loaded for STW and LDW instructions, MAR[0] == 1, then the exception should occur. If CLD_MAR0 is high and MAR[0] == 1, then Jbits 5, 3, 2, and 1 are set to high, and the state machine branches to 63 assuming bits 4 and 0 are high.

PSR[15]:
PSR[15] is used in state 53 to determine whether or not the machine is already in SUPERVISOR mode, meaning that the PSR and USP (R6) don’t have to be staved, and that SSP doesn’t have to be loaded into the Stack Pointer (R6).

NEW CONTROL SIGNAL BITS

```
Gate_PSR      - Gates PSR[15] and Condition Codes to BUS
Gate_USP      - Gates USP to BUS
Gate_SSP      - Gates SSP to BUS
Gate_SP       - Gates in/decremented stack pointer (R6)
Gate_IE_V     - Gate the interrupt vector table address to BUS
Gate_Dec_PC   - Gates the decremented (by 2) PC to BUS
LD_USP        - Enable USP to load the stack pointer (R6)
LD_SSP        - Enable SSP to load stack pointer (R6)
LD_PSR        - Enable PSR to load bit 15 of the BUS
              - Note: Control must also assert LD.CC when loading PSR
LD_TEMP_V     - Load temp register with interrupt/exception offset
SP_SEL        - Select stack pointer to be output from SR1OUT
SP_SELD       - Select stack pointer to be written to in REGFILE
DI_SEL        - Select whether to increment or decrement stack pointer by 2
VMUX1         - Select bits determining which INT/EXC offset to output
VMUX0         -
TM_COMP_MUX   - Selects whether to compare timer or MAR (BUS)
PSR_PRIV_MODE - Select whether to make PSR USER or SUPERVISOR mode
PSR_MDR_MODE  - SELECT whether to load PSR with data from BUS or MODE selecter
CLD_E1        - Load signal bit for Protected Exception to uSequencer
CLD_E2        - Load signal bit for Timer Interrupt to uSequencer
CLD_E3        - Load signal bit for Protected Exception (TRAP specific) to uSequencer
CLD_MAR0      - Load signal bit for Unaligned Access Exception to uSequencer
CLD_PSR       - Load signal bit for determining USER/SUPERVISOR mode
```

Extra Notes:

1. The logic that checks for E1 and E3 adds to the critical path length of the machine. This could easily be avoided by adding an extra state in between the states where E1 and E3 are checked and their subsequent states, and changing the BUS input of the comparator inputs to MAR. However, this would decrease the number of available states from 8 to 5, and I have been told by TA's that we will need at least 8 states to implement changes in later labs. So, for now, the cycle length will have to be lengthened.
    1. The above paragraph explains my original reasoning before code submission. I have realized that I could have wired the Base+offset outputs for each of the load/stores and trap states, anded them with new CLD_Enables micro control bits, and wired those outputs to the or gates of the J-bits in the microsequencer. This would remove the longer critical path length introduced by wiring the BUS to the microsequencer logic. However, this would add several control bits to the microinstructions and be different than what I implemented in lc3bsim4.c and ucode4, and I will not include the changes in this documentation.
2. Regarding the code:
    1. The next microinstruction bits are now changed both in eval_microsequencer and latch_values. This is because E1 and E3 depend on the BUS, which isn't driven properly until the latch_values function is called in the simulator. (The critical path length is even lengthened in the simulator).
    2. There is a TIE global variable used in the simulator code to determine whether or not a timer interrupt will be allowed to execute and, in this lab, limits the number of interrupts to 1. This was not included in the Data Path because I believe the interrupt service routine should acknowledge that it has occurred, set the new Int_Cycle value (Cycle_Count of next interrupt), and decide whether or not to re-arm the timer interrupt (which could be some memory location that feeds to the microsequencer logic). However, this is not how it was implemented in this programming lab.