CS152/252A
Spring 2026

Architectures, Microcoding

Discussion 1

1 Q1. Architecture vs Microarchitecture

True or false: The following is architecturally visible (exposed by the architecture)?

(a) Register file entries in a classical RISC pipeline

(b) The stack in a stack architecture

(c) Pipeline registers

(d) Branch-delay / load-delay slots

(e) NOPs

(f) Pipeline bubbles

(g) Condition codes, status flags

(h) Memory address width

(i) Instruction/data caches

2 Q2. Microcoded vs Pipelined

How does a microcoded machine differ from a classic RISC pipeline?

Why is a simpler microarchitecture generally possible with microcoding?

3 Q3. Microprogramming

Implement a conditional memory-to-memory move instruction in microcode for the single-bus RISC-V machine described in Handout 1 The instruction has the following format:

`CMOVM (rd), (rs1), rs2`

CMOVM performs the following operation: If the value in rs2 is true (non-zero), then the memory word loaded from the address in rs1 is stored to the address in rd.

```
if R[rs2] != 0
  M[rd] := M[rs1]
```

Fill in the following table with the microinstructions and control signals. Optimize your microprogram to minimize the number of cycles and to set entries to don't-cares (*) wherever possible.

State	Pseudocode	IR Ld	Reg Sel	Reg Wr	Reg En	A Ld	B Ld	ALUOp	ALU En	MA Ld	Mem Wr	Mem En	Imm Sel	Imm En	μBr	Next State
FETCH0	MA := PC; A := PC	*	PC	0	1	1	*	*	0	1	0	0	*	0	N	*
	IR := Mem	1	*	0	0	0	*	*	0	0	0	1	*	0	S	*
	PC := A + 4	0	PC	1	0	0	*	INC_A_4	1	*	0	0	*	0	D	*
...																
NOP0	μBr to FETCH0	*	*	0	0	*	*	*	0	*	0	0	*	0	J	FETCH0
CMOVM0:																

Appendix A. A Cheat Sheet for the Bus-based RISC-V Implementation

For your reference, we have reproduced the bus-based datapath diagram as well as a summary of some important information about microprogramming in the bus-based architecture.

Remember that you can use the following ALU operations:

ALUOp	ALU Result Output
COPY_A	A
COPY_B	B
INC_A_1	A+1
DEC_A_1	A-1
INC_A_4	A+4
DEC_A_4	A-4
ADD	A+B
SUB	A-B
SLT	Signed(A) < Signed(B)
SLTU	A < B

Table H1-2: Available ALU operations

Also remember that μBr (*micro*branch) column in Table H1-3 represents a 3-bit field with six possible values: N, J, EZ, NZ, D, and S. If μBr is N (next), then the next state is simply (*current state* + 1). If it is J (jump), then the next state is *unconditionally* the state specified in the Next State column (i.e., an unconditional microbranch). If it is EZ (branch-if-equal-zero), then the next state depends on the value of the ALU's *zero* output signal (i.e., a conditional microbranch). If *zero* is asserted (== 1), then the next state is that specified in the Next State column, otherwise, it is (*current state* + 1). NZ (branch-if-not-zero) behaves exactly like EZ, but instead performs a microbranch if *zero* is not asserted (!= 1). If μBr is D (dispatch), then the FSM looks at the opcode and function fields in the IR and goes into the corresponding state. If S, the μPC spins if *busy?* is asserted, otherwise goes to (*current state* +1).