[Updated 11/06/2017]

ISA for the CS 3220 Project

```
Opcodes are 4 bits.
Functions (secondary opcodes) are 4 bits.
Immediate operands are 16 bits.
Register indices are 4 bits (RS1, RS2, RD).
```

Rough Verilog example:

```
wire[31:0] iword;
wire[3:0]  rd, rs1, rs2
wire[15:0] imm;
wire[3:0]  fn;
wire[3:0]  opcode;

assign fn     = iword[31:28];
assign opcode = iword[27:24];
assign imm    = iword[23:8];
assign rs2    = iword[11:8];
assign rs1    = iword[7:4];
assign rd     = iword[3:0];
```

ALU-R

```
ADD   : {fmt: "RS2,RS1,RD",iword: "0011 1111 000000000000 RS2 RS1 RD"}
SUB   : {fmt: "RS2,RS1,RD",iword: "0010 1111 000000000000 RS2 RS1 RD"}
AND   : {fmt: "RS2,RS1,RD",iword: "0111 1111 000000000000 RS2 RS1 RD"}
OR    : {fmt: "RS2,RS1,RD",iword: "0110 1111 000000000000 RS2 RS1 RD"}
XOR   : {fmt: "RS2,RS1,RD",iword: "0101 1111 000000000000 RS2 RS1 RD"}
NAND  : {fmt: "RS2,RS1,RD",iword: "1011 1111 000000000000 RS2 RS1 RD"}
NOR   : {fmt: "RS2,RS1,RD",iword: "1010 1111 000000000000 RS2 RS1 RD"}
XNOR  : {fmt: "RS2,RS1,RD",iword: "1001 1111 000000000000 RS2 RS1 RD"}
```

ALU-I

```
ADDI  : {fmt: "imm,RS1,RD",iword: "0011 1011 imm[15:0] RS1 RD"}
SUBI  : {fmt: "imm,RS1,RD",iword: "0010 1011 imm[15:0] RS1 RD"}
ANDI  : {fmt: "imm,RS1,RD",iword: "0111 1011 imm[15:0] RS1 RD"}
ORI   : {fmt: "imm,RS1,RD",iword: "0110 1011 imm[15:0] RS1 RD"}
XORI  : {fmt: "imm,RS1,RD",iword: "0101 1011 imm[15:0] RS1 RD"}
NANDI : {fmt: "imm,RS1,RD",iword: "1011 1011 imm[15:0] RS1 RD"}
NORI  : {fmt: "imm,RS1,RD",iword: "1010 1011 imm[15:0] RS1 RD"}
XNORI : {fmt: "imm,RS1,RD",iword: "1001 1011 imm[15:0] RS1 RD"}
MVHI  : {fmt: "imm,RD",    iword: "1111 1011 imm[15:0] 0000 RD"}
```

Load/Store

```
LW    : {fmt: "imm(RS1),RD",  iword: "0000 1001 imm[15:0]  RS1 RD "}
SW    : {fmt: "imm(RS1),RS2", iword: "0000 1000 imm[15:0]  RS2 RS1"}
```

CMP-R

```
F     : {fmt: "RS2,RS1,RD", iword: "0011 1110 000000000000 RS2 RS1 RD"}
EQ    : {fmt: "RS2,RS1,RD", iword: "1100 1110 000000000000 RS2 RS1 RD"}
LT    : {fmt: "RS2,RS1,RD", iword: "1101 1110 000000000000 RS2 RS1 RD"}
LTE   : {fmt: "RS2,RS1,RD", iword: "0010 1110 000000000000 RS2 RS1 RD"}
T     : {fmt: "RS2,RS1,RD", iword: "1111 1110 000000000000 RS2 RS1 RD"}
NE    : {fmt: "RS2,RS1,RD", iword: "0000 1110 000000000000 RS2 RS1 RD"}
GTE   : {fmt: "RS2,RS1,RD", iword: "0001 1110 000000000000 RS2 RS1 RD"}
GT    : {fmt: "RS2,RS1,RD", iword: "1110 1110 000000000000 RS2 RS1 RD"}
```

CMP-I

```
FI    : {fmt: "imm,RS1,RD", iword: "0011 1010 imm[15:0] RS1 RD"}
EQI   : {fmt: "imm,RS1,RD", iword: "1100 1010 imm[15:0] RS1 RD"}
LTI   : {fmt: "imm,RS1,RD", iword: "1101 1010 imm[15:0] RS1 RD"}
LTEI  : {fmt: "imm,RS1,RD", iword: "0010 1010 imm[15:0] RS1 RD"}
TI    : {fmt: "imm,RS1,RD", iword: "1111 1010 imm[15:0] RS1 RD"}
NEI   : {fmt: "imm,RS1,RD", iword: "0000 1010 imm[15:0] RS1 RD"}
GTEI  : {fmt: "imm,RS1,RD", iword: "0001 1010 imm[15:0] RS1 RD"}
GTI   : {fmt: "imm,RS1,RD", iword: "1110 1010 imm[15:0] RS1 RD"}
```

BRANCH

```
BF    : {fmt: "imm,RS2,RS1", iword: "0011 0000 imm[15:0] RS2 RS1"}
BEQ   : {fmt: "imm,RS2,RS1", iword: "1100 0000 imm[15:0] RS2 RS1"}
BLT   : {fmt: "imm,RS2,RS1", iword: "1101 0000 imm[15:0] RS2 RS1"}
BLTE  : {fmt: "imm,RS2,RS1", iword: "0010 0000 imm[15:0] RS2 RS1"}
BEQZ  : {fmt: "imm,RS1",     iword: "1000 0000 imm[15:0] 0000 RS1"}
BLTZ  : {fmt: "imm,RS1",     iword: "1001 0000 imm[15:0] 0000 RS1"}
BLTEZ : {fmt: "imm,RS1",     iword: "0110 0000 imm[15:0] 0000 RS1"}

BT    : {fmt: "imm,RS2,RS1", iword: "1111 0000 imm[15:0] RS2 RS1"}
BNE   : {fmt: "imm,RS2,RS1", iword: "0000 0000 imm[15:0] RS2 RS1"}
BGTE  : {fmt: "imm,RS2,RS1", iword: "0001 0000 imm[15:0] RS2 RS1"}
BGT   : {fmt: "imm,RS2,RS1", iword: "1110 0000 imm[15:0] RS2 RS1"}
BNEZ  : {fmt: "imm,RS1",     iword: "0100 0000 imm[15:0] 0000 RS1"}
BGTEZ : {fmt: "imm,RS1",     iword: "0101 0000 imm[15:0] 0000 RS1"}
BGTZ  : {fmt: "imm,RS1",     iword: "1010 0000 imm[15:0] 0000 RS1"}

JAL   : {fmt: "imm(RS1),RD", iword: "0000 0001 imm[15:0] RS1 RD"}
```

PSEUDO INSTRS

```
B is implemented using BEQ
BR    : {fmt: "imm",          itext: ["BEQ  imm,R6,R6"]}

NOT is implemented using NAND
NOT   : {fmt: "RS,RD",        itext: ["NAND RS,RS,RD"]}

BLE, BGE are implemented using LTE/GTE and BNEZ
BLE   : {fmt: "imm,RS2,RS1",  itext: ["LTE  RS2,RS1,R6","BNEZ imm,R6"]}
BGE   : {fmt: "imm,RS2,RS1",  itext: ["GTE  RS2,RS1,R6","BNEZ imm,R6"]}

CALL/RET/JMP are implemented using JAL
CALL  : {fmt: "imm(RS1)",     itext: ["JAL  imm(RS1),RA"]}
```

```
RET   : {fmt: "",             itext: ["JAL  0(RA),R9"]}
JMP   : {fmt: "imm(RS1)",     itext: ["JAL  imm(RS1),R9"]}
```

Single cycle processor specification

Requirements for the processor:

- It must implement the documented ISA.
- PC must start at (byte address) 0x40.
- SW to address 0xF0000000 must display bits 15 to 0 as hexadecimal digits on the HEX display.
- SW to address 0xF0000004 must display bits 9 to 0 on LEDR.
- LW from address 0xF0000010 must read the current KEY state. The result should be 0 when no KEY pressed, and 0xF when all are pressed.
- LW from address 0xF0000014 reads SW state.
- The 32-bit value we read should really be {22'b0,SWd) where SWd is a debounced value of SW.

Assembler specification

The assembler must read an assembly file containing a program that follows the ISA spec, and it must output a MIF file with 2048 32-bit words of memory (8192 bytes in total).

Opcode and function mappings

For the following tables, the rows indices represent the most significant bits (MSB) and the column indices are the LSB.

General opcode mapping

LSB / MSB	**00**	**01**	**10**	**11**
00	BRANCH	JAL		
01				
10	SW	LW	CMP-I	ALU-I
11			CMP-R	ALU-R

ALU-R/ALU-I function mapping

LSB / MSB	**00**	**01**	**10**	**11**
00			SUB/SUBI	ADD/ADDI
01		XOR/XORI	OR/ORI	AND/ANDI
10		XNOR/XNORI	NOR/NORI	NAND/NANDI
11				MVHI

CMP-R/CMP-I function mapping

MSB \ LSB	00	01	10	11
00	NE/NEI	GTE/GTEI	LTE/LTEI	F/FI
01				
10				
11	EQ/EQI	LT/LTI	GT/GTI	T/TI

BRANCH function mapping

MSB \ LSB	00	01	10	11
00	BNE	BGTE	BLTE	BF
01	BNEZ	BGTEZ	BLTEZ	
10	BEQZ	BLTZ	BGTZ	
11	BEQ	BLT	BGT	BT

Instruction format

- ALU-R
  - rd = rs1 op rs2
- CMP-R
  - rd = (rs1 op rs2)? 1: 0
- Store
  - Mem[rs1 + signextension(imm)] = rs2
- Load
  - rd = Mem[rs1 + signextension(imm)]
- ALU-I
  - rd = rs1 op signextension(imm)
- CMP-I
  - rd = (rs1 op signextension(imm)) ? 1 : 0
- BRANCH
  - if (rs1 op rs2) PC = PC + 4 + (signextension(imm) * 4)
- JAL
  - rd = PC + 4
  - PC = rs1 + 4 * signextension(imm)

Assembler syntax

- Instruction opcodes and register names:
  - Are reserved words (can't be used as labels).
  - Appear in either lowercase or uppercase.
  - If there is a destination register, it is listed first.
- Labels:
  - Are created using a name and then ":" at the start of a line
  - Corresponds to the address where label created
- Immediate operands – number or label
  - If number, hex (C format, e.g. 0xffff) or decimal (can have - sign)
  - If label, just use the name of the lable (without ":")
    - For PC-relative, the immediate field is label_addr-PC-4
    - For other insts, the immediate field is 16 least-significant bits of label_addr
- Each register has multiple names:
  - R0..R3 are also A0..A3 (function arguments, caller saved)
  - R3 is also RV (return value, caller saved)
  - R4..R5 are also T0..T1 (temporaries, caller saved)
  - R6..R8 are also S0..S2 (callee-saved values)
  - R9 reserved for assembler use
  - R10..R11 reserved for system use (we'll see later for what)
  - R12 is GP (global pointer)
  - R13 is FP (frame pointer)
  - R14 is SP (stack pointer)
    - Stack grows down, SP points to lowest in-use address
  - R15 is RA (return address)

Special assembler instructions

- .ORIG <number>
  - Changes "current" address to <number>
- .WORD <value>
  - Places 32-bit word <value> at the current address
  - <value> can be a number or a label name
  - If label name, value is the full 32-bit label_addr
- .NAME <name>=<value>
  - Defines a name (label) with a given value (number)