ECE 449:
Instruction Set (16-bit)

In this lab, you will design and implement a pipelined 16-bit processor on FPGA. The instructions set of the processor is as follows:

Instructions Set

We use a RISC-like instruction set in the project. Instructions are 1-word (16-bits) long and are word aligned. There are eight 16-bit general purpose registers; R_0, R_1, … R_7. R_7 has also some additional special roles. It acts as a link register to receive the program counter for the branch-to-subroutine (procedure) instruction. It is also the target for the load-immediate instruction. The memory address space is 2^{16}= 65,536 bytes and it is byte addressable.

For extra credit, there are some optional instructions. These optional instructions work with the stack. PUSH and POP instructions write and read to/from the stack. A dedicated register, stack pointer (SP), points to the top of the stack. Also, interrupt is optional in the project. When interrupt occurs, the address of the instruction next to the interrupted instruction is saved on top of the stack, and the PC is loaded with data in address 1 of memory. To return-from-interrupt instruction (RTI), loads the PC with the top of stack, and the flow of the program resumes from the instruction after the interrupted instruction.

There are three different instruction formats:

1) A-Format

Figure 1 depicts a-format instructions. These instructions are 2-byte long. The Op-code is the 7 most significant bits i.e. bits 15 to 9 while the remaining 9 bits are divided into three fields that determine operand registers.

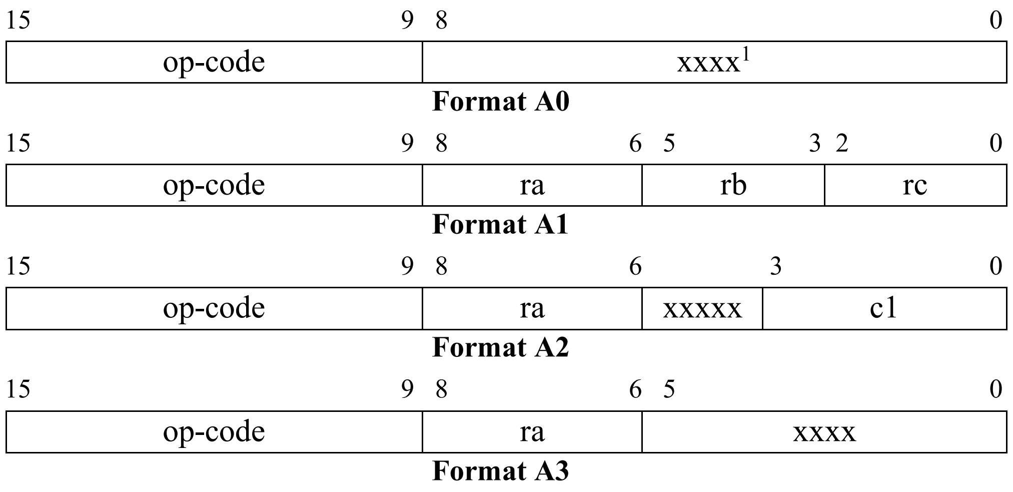

Figure 1. A-format instructions.

Table I shows the op-code values for a-format instructions and explains their

[1] xxxx indicates that the field is not used (don’t care)

functionality. R[ra] indicates value of register *ra.* A special TEST instruction determines whether a specific register's contents are zero, positive or negative. Please note that there is no functionality that stores or restores the contents of the zero (Z) or negative (N) flags explicitly. However, in the project extension, interrupt handling requires the saving and restoring the processor state including the N and Z flags. . The TEST instruction, is meant to be issued before a conditional branch to set the conditions for this branch. The processor has a 16-bit input port and a 16-bit output port. The ports of the processor are connected to external pins. IN, and OUT instructions transfer values between the processor ports and the internal registers.
As an example, the *ADD r3, r2, r1* instruction, has the following *op-code*, *ra*, *rb* and *rc* fields:
op-code = 1
ra = 3
rb = 2
rc=1

The bit stream for the instruction is: 0000001011010001. Hence, the hexadecimal format of the instruction is: 0x02D1

Table I. A-format instructions.

Mne-monic	Op-code	Function	Type	Syntax
NOP	0	Nothing	A0	*NOP*
ADD	1	R[ra] ← R[rb] + R[rc];	A1	*ADD ra,rb,rc*
SUB	2	R[ra] ← R[rb] – R[rc];	A1	*SUB ra,rb,rc*
MUL	3	R[ra] ← R[rb] × R[rc];	A1	*MUL ra,rb,rc*
NAND	4	R[ra] ← R[ra] NAND R[rb]:	A1	*NAND ra,rb,rc*
SHL	5	n:=c1<3…0>): (n|0) → (R[ra]<15…0>←R[ra]<15-n…0>#(n@0)): (n=0) → () ;	A2	*SHL ra#n*
SHR	6	n:=c1<3…0>): (n|0) → (R[ra]<15…0>←(n@0)#R[ra]<15…n>): (n=0) → () ;	A2	*SHR ra#n*
TEST	7	(R[ra) = 0) →Z ← 1; else →Z ← 0: (R[ra] < 0) →N ← 1; else →N ← 0;	A3	*TEST ra*
OUT	32	OUT.PORT ← R[ra];	A3	*OUT ra*
IN	33	R[ra] ← IN.PORT;	A3	*IN ra*

2) B-Format

B-format instructions are are used for branch instructions. As figure 2 shows, the B-format determines a (relative) displacement and for the second format, a register. The .

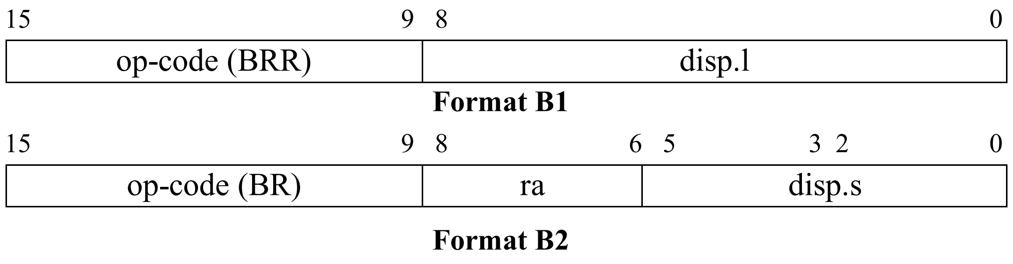

Figure 2. B-format (Branch) instructions.

Table II. B-format instructions.

Mne-monic	Op-code	Function	Type	Syntax
BRR	64	PC ← PC[†]+2*disp.l {sign extended 2's complement}	B1	*BRR +disp.l*
BRR.N	65	(N=1) → PC ← PC+2*disp.l {sign extended 2's complement}; (N=0) → PC ← PC+2 { 2's complement};	B1	*BRR.N +disp.l*
BRR.Z	66	(Z=1) → PC ← PC+2*disp.l {sign extended 2's complement}; (Z=0) → PC ← PC+2 { 2's complement};	B1	*BRR.Z +disp.l*
BR	67	PC ← R[ra]+2*disp.s {sign extended 2's complement}	B2	*BR ra+disp.s*
BR.N	68	(N=1) → PC ← R[ra] {word aligned}+2*disp.s {sign extended 2's complement}; (N=0) → PC ← PC+2 { 2's complement};	B2	*BR.N ra+disp.s*
BR.Z	69	(Z=1) → PC ← R[ra] {word aligned}+2*disp.s {sign extended 2's complement}; (Z=0) → PC ← PC+2 { 2's complement};	B2	*BR.Z ra+disp.s*
BR.SUB	70	r7 ← PC + 2; PC ← R[ra] {word aligned}+2*disp.s {sign extended 2's complement};	B2	*BR.SUB ra+disp.s*
RETURN	71	PC ← r7;	A0	*RETURN*

Table II shows details of the B-format instructions i.e. branch instructions. There are two different types. The first is branch relative (BRR). This indicates that the branch is relative to the program counter (PC). The branch can be forward when the displacement is positive or backwards when the displacement is negative. Note the two's complement arithmetic and the fact that the displacement is always multiplied by two to ensure word (two-byte) alignment.

The alternate type, is branch absolute (BR). In this case, the base target address is stored in one of the registers while the target address is formed when the displacement (multiplied by two) is added to the base address. The base address is assumed to be word aligned i.e. the least significant bit is zero[2].

BR.Z(N) are conditional branches. If the Z(N) flag is one, the branch is taken otherwise, it is not. BR.SUB is used for a subroutine call. It saves the address of the next instruction into register *r7* and branches to the address of subroutine formed by the addition of the contents of the specified register plus the stated displacement. At the end of subroutine, a RETURN instruction copies the contents of *r7* into the PC.

Note that branching to a subroutine, does not save any state except the program counter.

3) L-Format

L-format instructions are used for loads, stores and moves. These are two-operand instructions defining a source and a destination. There are of two types: immediate and to/from register. The immediate load, loads one (immediate) byte to the MSB or LSB part of register r7. The immediate loads are used to formulate effective addresses in *r7*. This introduces an asymmetry in the ISA, but such a format was used to ensure that all instructions are of the same length (i.e. 2 bytes) and therefore simplify (and hence speed-up) the design of the instruction pipeline.

The to/from register format defines the register holding the effective address and the register that will receive the data from memory or holds the data to be moved to memory

[†] For the PC operations, the argument is the value of the PC just before the instruction is fetched, while the result is the value of the PC at the conclusion of the execution of the instruction.

[2] To ensure word alignment, you mask the LSB of R[ra]

(or to another register in the case of MOVE).
are two bytes and are used for load/store instructions. The first byte holds op-code and ra and the second byte holds address of memory or an immediate value. Figure 3 shows the L-format instructions. Note that in L-format instructions, the first three low order bits of the first byte are unused.

15 9	8	7 0
op-code	m.l	imm

Format L.1

15 9	8 6	5 3	2 0
op-code	r.dest	r.src	

Format L.2

Figure 3. L Format instructions

Table III shows details of L-format instructions. LOAD and STORE instructions write/read register *ra* into/from address ea. M[ea] shows the content of memory with address ea. LOADIMM writes a constant value (imm) into register ra.

Table III. L-format instructions.

Mnemonic	Op-code	Function	Type	Syntax
LOAD	16	R[r.dest] ← M[R[r.src]];	L2	*LOAD r.dest, r.src* *LOAD r.drst, @r.src*
STORE	17	M[R[r.dest]] ← R[r.src];	L2	*STORE r.dest, r.src* *STORE @r.dest, r.src*
LOADIMM	18	(m.l=1) →R7〈15⋯8〉← imm: (m.l=0) →R7〈7⋯0〉← imm;	L1	*LOADIMM.upper #n* *LOADIMM.lower #n*
MOV	19	R[r.dest] ←R[r.src];	L2	*MOV dest,src*

4) Optional instructions

In this section, we discuss instructions that are not mandatory to implement for the project. Table IV shows the optional instructions. PUSH and POP are similar to a-format instructions. In PUSH, ra is 1 and in POP, ra is 0. SP is a special register which points to the top of the stack. The initial value of the SP is to be loaded using the instruction LOAD.SP. Normally the top of the stack is the highest address of the available memory in the system . The PUSH instruction writes register ra into memory address pointed by SP. Then SP is decremented. The POP instruction, first increments SP, and then, data pointed by SP is written into ra.
Please note that this specification does not include checking of whether the stack pointer exceeds the available memory. It is assumed that this will be handled by the memory controller.
The processor has a dedicated pin for external interrupt. On the rising edge of the interrupt pin, the address of the next instruction is written onto the stack and the SP is decremented (M[SP--] ← PC). The PC is loaded with address 1 (PC ← M[1]), and the processor jumps into interrupt service routine. Z and N flag are written into memory. At the end of interrupt service routine, the RTI instruction executes. RTI is an A0-format instruction. It increments the SP and loads the PC with the top of the stack. Z and N flags are restored from memory.

Table IV. Optional instructions.

Mne-	Op-	Function		

monic	code			
PUSH	96	M[SP--] ← R[ra];	A3	*PUSH ra*
POP	97	R[rb] ← M[++SP];	A3	*POP ra*
LOAD.SP	98	SP←R[ra];	A3	*LOAD.SP ra*
RTI	99	PC ← M[++SP]: {Z,N} restored	A0	*RTI*

Table V shows summary of all instructions. Instructions in italic format are optional.

Table V. Summary of all instructions.

Mne-monic	Op-code	Function	Type	Syntax
NOP	0	Nothing	A0	*NOP*
ADD	1	R[ra] ← R[rb] + R[rc];	A1	*ADD ra,rab,rc*
SUB	2	R[ra] ← R[rb] – R[rc];	A1	*SUB ra,rb,rc*
MUL	3	R[ra] ← R[rb] × R[rc];	A1	*MUL ra,rb,rc*
NAND	4	R[ra] ← R[ra] NAND R[rb]:	A1	*NAND ra,rb,rc*

SHL	5	n:=c1<3…0>):
		(n|0) → (R[ra]<15…0>←R[ra]<15-n…0>#(n@0)):
		(n=0) → () ;
			A2	*SHL ra#n*

SHR	6	n:=c1<3…0>):
		(n|0) → (R[ra]<15…0>←(n@0)#R[ra]<15…n>):
		(n=0) → () ;
			A2	*SHR ra#n*

TEST	7	(R[ra) = 0) →Z ← 1; else →Z ← 0: (R[ra] < 0) →N ← 1; else →N ← 0;	A3	*TEST ra*
OUT	32	OUT.PORT ← R[ra];	A3	*OUT ra*
IN	33	R[ra] ← IN.PORT;	A3	*IN ra*
BRR	64	PC ← PC[†]+2*disp.l {sign extended 2's complement}	B1	*BRR +disp.l*

BRR.N	65	(N=1) → PC ← PC+2*disp.l {sign extended 2's complement};
		(N=0) → PC ← PC+2 { 2's complement};
			B1	*BRR.N +disp.l*

BRR.Z	66	(Z=1) → PC ← PC+2*disp.l {sign extended 2's complement};
		(Z=0) → PC ← PC+2 { 2's complement};
			B1	*BRR.Z +disp.l*

BR	67	PC ← R[ra]+2*disp.s {sign extended 2's complement}	B2	*BR ra+disp.s*

BR.N	68	(N=1) → PC ← R[ra] {word aligned}+2*disp.s {sign extended 2's complement};
		(N=0) → PC ← PC+2 { 2's complement};
			B2	*BR.N ra+disp.s*

BR.Z	69	(Z=1) → PC ← R[ra] {word aligned}+2*disp.s {sign extended 2's complement};
		(Z=0) → PC ← PC+2 { 2's complement};
			B2	*BR.Z ra+disp.s*

BR.SUB	70	r7 ← PC + 2; PC ← R[ra] {word aligned}+2*disp.s {sign extended 2's complement};	B2	*BR.SUB ra+disp.s*
RETURN	71	PC ← r7;	A0	*RETURN*
LOAD	16	R[r.dest] ← M[R[r.src]];	L2	*LOAD r.dest, r.src* *LOAD r.drst, @r.src*
STORE	17	M[R[r.dest]] ← R[r.src];	L2	*STORE r.dest, r.src* *STORE @r.dest, r.src*
LOADIMM	18	(m.l=1) →R7<15⋯8>← imm: (m.l=0) →R7<7⋯0>← imm;	L1	*LOADIMM.upper #n* *LOADIMM.lower #n*
MOV	19	R[r.dest] ←R[r.src];	L2	*MOV dest,src*
PUSH	96	M[SP--] ← R[ra];	A3	*PUSH ra*
POP	97	R[rb] ← M[++SP];	A3	*POP ra*
LOAD.SP	98	SP←R[ra];	A3	*LOAD.SP ra*
RTI	99	PC ← M[++SP]: {Z,N} restored	A0	*RTI*

[†] For the PC operations, the argument is the value of the PC just before the instruction is fetched, while the result is the value of the PC at the conclusion of the execution of the instruction.

EVALUATION CRITERIA

Each group shall normally comprise two members. In special circumstances, and with the permission of the lab demonstrator, a group may comprise three members.

Preliminary Design Review (PDR)

Each group shall present a preliminary design during a Preliminary Design Review (PDR). This will consist of a short report and a 5-10 minute presentation.
The PDR shall include the concrete specification of the ISA and the block-level decomposition and specification of the data and control paths.
The preliminary report and presentation shall count for 20% of the Lab mark.

Final Design Review and Project Demonstration

Each group shall present the final design and submit a final report during the Final Design Review (last day of classes).
The Final Design Report and the Presentation shall count for 35% of the Lab mark.
Each group shall demonstrate its design during the Final Design Review. The demonstration shall count for 45% of the Lab mark.