EECS 427 RISC PROCESSOR

ISA FOR EECS 427 PROCESSOR

				ImmHi/	ImmLo/	
		OP Code	Rdest	OP Code Ext	Rsrc	
Mnemonic	Operands	15-12	11-8	7-4	3-0	Notes (* is Baseline)
ADD	Rsrc, Rdest	0000	Rdest	0101	Rsrc	*
ADDI	Imm, Rdest	0101	Rdest	ImmHi	ImmLo	* Sign extended Imm
ADDU	Rsrc, Rdest	0000	Rdest	0110	Rsrc	
ADDUI	Imm, Rdest	0110	Rdest	ImmHi	ImmLo	Sign extended Imm
ADDC	Rsrc, Rdest	0000	Rdest	0111	Rsrc	
ADDCI	Imm, Rdest	0111	Rdest	ImmHi	ImmLo	Sign extended Imm
MUL	Rsrc, Rdest	0000	Rdest	1110	Rsrc	
MULI	Imm, Rdest	1110	Rdest	ImmHi	ImmLo	Sign extended Imm
SUB	Rsrc, Rdest	0000	Rdest	1001	Rsrc	*
SUBI	Imm, Rdest	1001	Rdest	ImmHi	ImmLo	* Sign extended Imm
SUBC	Rsrc, Rdest	0000	Rdest	1010	Rsrc	
SUBCI	Imm, Rdest	1010	Rdest	ImmHi	ImmLo	Sign extended Imm
CMP	Rsrc, Rdest	0000	Rdest	1011	Rsrc	*
CMPI	Imm, Rdest	1011	Rdest	ImmHi	ImmLo	* Sign extended Imm
AND	Rsrc, Rdest	0000	Rdest	0001	Rsrc	*
ANDI	Imm, Rdest	0001	Rdest	ImmHi	ImmLo	* Zero extended Imm
OR	Rsrc, Rdest	0000	Rdest	0010	Rsrc	* NOP=OR R0,R0
ORI	Imm, Rdest	0010	Rdest	ImmHi	ImmLo	* Zero extended Imm
XOR	Rsrc, Rdest	0000	Rdest	0011	Rsrc	*
XORI	Imm, Rdest	0011	Rdest	ImmHi	ImmLo	* Zero extended Imm
MOV	Rsrc, Rdest	0000	Rdest	1101	Rsrc	*
MOVI	Imm, Rdest	1101	Rdest	ImmHi	ImmLo	* Zero extended Imm
LSH	Ramount, Rdest	1000	Rdest	0100	Ramount	* -15 to 15 (2s compl)
LSHI	Imm, Rdest	1000	Rdest	000s	ImmLo	* s = sign (0=left, 2s comp)
ASHU	Ramount, Rdest	1000	Rdest	0110	Ramount	-15 to 15 (2s comp)
ASHUI	Imm, Rdest	1000	Rdest	001s	ImmLo	s = sign (0=left, 2s comp)
LUI	Imm, Rdest	1111	Rdest	ImmHi	ImmLo	* Load & 8 bit Left Shift

ISA FOR EECS 427 PROCESSOR

LOAD	Rdest, Raddr	0100	Rdest	0000	Raddr	*
STOR	Rsrc, Raddr	0100	Rsrc	0100	Raddr	*
SNXB	Rsrc, Rdest	0100	Rdest	0010	Rsrc	
ZRXB	Rsrc, Rdest	0100	Rdest	0110	Rsrc	
Scond	Rdest	0100	Rdest	1101	cond	
Bcond	disp	1100	cond	DispHi	DispLo	* 2s comp displacement
Jcond	Rtarget	0100	cond	1100	Rtarget	*
JAL	Rlink, Rtarget	0100	Rlink	1000	Rtarget	*
TBIT	Roffset, Rsrc	0100	Rsrc	1010	Roffset	Offset = 0 to 15
TBITI	Imm, Rsrc	0100	Rsrc	1110	Offset	Offset = 0 to 15
LPR	Rsrc, Rproc	0100	Rsrc	0001	Rproc	
SPR	Rproc, Rdest	0100	Rproc	0101	Rdest	
DI		0100	0000	0011	0000	
EI		0100	0000	0111	0000	
EXCP	vector	0100	0000	1011	vector	
RETX		0100	0000	1001	0000	
WAIT		0000	0000	0000	0000	
Unused OP code		0000		0100		
Unused OP code		0000		1000		
Unused OP code		0000		1100		
Unused OP code		0000		1111		
Unused OP code		0100		1111		
Unused OP code		1000		0101		
Unused OP code		1000		0111		
Unused OP code		1000		1xxx		

ISA FOR EECS 427 PROCESSOR

Opcodes, Extended Opcodes and Condition Codes

OP Code

Bits	13,12			
15,14	00	01	10	11
00	Register	ANDI	ORI	XORI
01	Special	ADDI	ADDUI	ADDCI
10	Shift	SUBI	SUBCI	CMPI
11	Bcond	MOVI	MULI	LUI

Shift

Bits	5,4			
7,6	00	01	10	11
00	LSHI	LSHI	ASHUI	ASHUI
01	LSH		ASHU	
10				
11				

Register

Bits	5,4			
7,6	00	01	10	11
00	WAIT	AND	OR	XOR
01		ADD	ADDU	ADDC
10		SUB	SUBC	CMP
11		MOV	MUL	

cond

Bits	9,8/1,0			
11,10/3,2	00	01	10	11
00	EQ	NE	CS	CC
01	HI	LS	GT	LE
10	FS	FC	LO	HS
11	LT	GE	UC	

Special

Bits	5,4			
7,6	00	01	10	11
00	LOAD	LPR	SNXB	DI
01	STOR	SPR	ZRXB	EI
10	JAL	RETX	TBIT	EXCP
11	Jcond	Scond	TBITI	

Blank entries are unused codes
Unshaded instructions are Baseline

EECS 427 RISC PROCESSOR

The group projects for EECS 427 will be based on the processor specification given in this document. The processor specification is based on RISC concepts and is implemented as a two stage pipeline. It uses a 16 bit word and address space, although for simplicity, each address refers to a complete word (two bytes), so the address space is 2^{17} bytes. All instructions are single word. Following the RISC approach, almost all instructions refer to a 16 entry register file. The highest nybble is the operation code, the next nybble is usually the destination register address, the remaining byte is an immediate data value for some instructions, or is split into a four bit operation code extension and a four bit source register address for other instructions. (A few instructions are different, so read the specifications carefully.) In order to make the project feasible for most groups in the available time, a “baseline” implementation is also given. This uses a selected subset of the instructions and the expectation is that implementation of the baseline processor is the minimum requirement for this course. Each group should plan a customization beyond the baseline, which makes the processor useful for a particular application. Typically, this may involve adding a few instructions beyond the baseline, special registers which allow certain operations to be done more efficiently, and an interface logic to external input/output devices. (Note: most projects are usually much closer to the baseline, than the full implementation.) All the baseline instructions should be implemented without change, so that your processor can execute a test program at the end of the term. Added instructions should normally use the “unused opcodes” which are listed in the Instruction Set Architecture (ISA). If it is necessary to replace some of the additional instructions (beyond the baseline), discuss it with the instructor. Because the register file can be physically large, it is acceptable to implement only eight of the sixteen registers (addresses 0 through 7) for the baseline processor, although the instruction format should not be changed.

A block diagram is given of the architecture of the baseline processor as a guideline, but you are free to make additions and changes to it, but you should still follow the RISC approach with pipelined stages. The following sections discuss the functions of the instructions. You should also refer to the notes in the list of instructions.

Notes on the Baseline Instruction Set

All ALU instructions (except CMP, CMPI - see below) write the result back to the destination register. Instructions ending with I are immediate and use the eight least-significant bits of the instruction as data, the others are direct, (i.e. instruction “*op* Rsrc/Imm, Rdest” performs

Rdest <-- Rdest *op* Imm (sign extended)

or

Rdest <-- Rdest *op* Rsrc

respectively).

For the baseline EECS 427 processor, the instructions marked with an asterisk in the instruction table should be implemented. Successive memory addresses can refer to 16 bit words instead of bytes. Of the baseline subset of instructions, the only ones which can change the program status register (PSR) are the arithmetic instructions ADD, ADDI, SUB,SUBI, CMP, CMPI. CMP and CMPI perform the same operations as SUB, SUBI but affect different PSR flags (see below) and do not write back the result. Only flags FLCNZ are needed for the baseline implementation.

ADD, ADDI, SUB,SUBI set the C flag if a carry/borrow from the most significant bit position occurs when the operands are treated as unsigned numbers, and set the F flag if an overflow occurs when the operands are treated as two’s complement numbers. (Note: the processor does not know which interpretation you are using, so must set both flags appropriately for each operation.) CMP,

CMPI perform a subtraction without write back to Rdest and set the Z flag if the result is zero, set the L flag if Rsrc/Imm > Rdest when the operands are treated as unsigned numbers (i.e. when a carry/borrow occurs), and set the N flag if Rsrc/Imm > Rdest when the operands are treated as two's complement numbers (N can be computed as the exclusive-or of L and the sign bits of Rsrc/Imm and Rdest). All other baseline instructions leave the flags unchanged.

Jcond, Bcond are absolute and relative jumps respectively based on the condition codes specified in the condition code (cond) table. (See Table 1.)

JAL (jump and link) stores the address of the next instruction in Rlink, and jumps to Rtarget. Its main use is for subroutine calls. Return with a JUC Rlink (where Rlink is the same register used to store the link).

LSH is a logical left shift by the number of bits specified in Rsrc/Imm treated as a signed twos complement number (which must be in the range -15 to +15). A negative left shift is effectively a right shift.

LOAD and STOR instructions load to, and store from the data memory location whose address is in register Raddr. The NOP instruction is really OR r0, r0 and does not need to be implemented separately. Unconditional jumps (JUMP) and branches (BR) are equivalent to JUC and BUC respectively, so do not need separate implementation either. Compilers may have these alternative instruction ops for convenience, however.

LUI (load upper immediate) loads the 8 bit immediate data into the upper (most significant) bits of the destination register.

MOV copies the source register or immediate into the destination register.

Notes on the Additional Instructions

In a full implementation, PSR (program status register) is a dedicated 16 bit register with flag entries (in the following order, MSB at the left) `rrrrIPE0NZF00LTC`, where the "r" entries are reserved, the "0" entries are zeros, I, E are used for interrupt processing, T, P are for program tracing (debugging), and the rest are flags have been defined elsewhere.

ADDU does the same as ADD but does not affect the PSR flags.

ADDC does the same as ADD except the C flag is also added in. It affects the same flags.

ASHU does an arithmetic left shift interpreting both operands as signed twos complement.

MUL multiplies: Rdest <-- Rsrc/Imm * Rdest. High order bits are truncated if they do not fit in Rdest. No flags are affected.

SUBC does the same as SUB except that the C flag is also subtracted. It affects the same flags.

SNXB converts the 8-bit operand in Rsrc to 16 bits in Rdest with sign-extension.

Scond sets Rdest = 1 if the condition is true (i.e. if the bit in the PSR is set), and resets Rdest = 0 if it is false (same condition codes as jump and branch instructions).

DI, EI, EXCP, RETX deal with interrupts and exceptions. Ask if you are interested in implementing any of them.

LPR, SPR load the PSR from Rsrc and store PSR into Rdest, respectively.

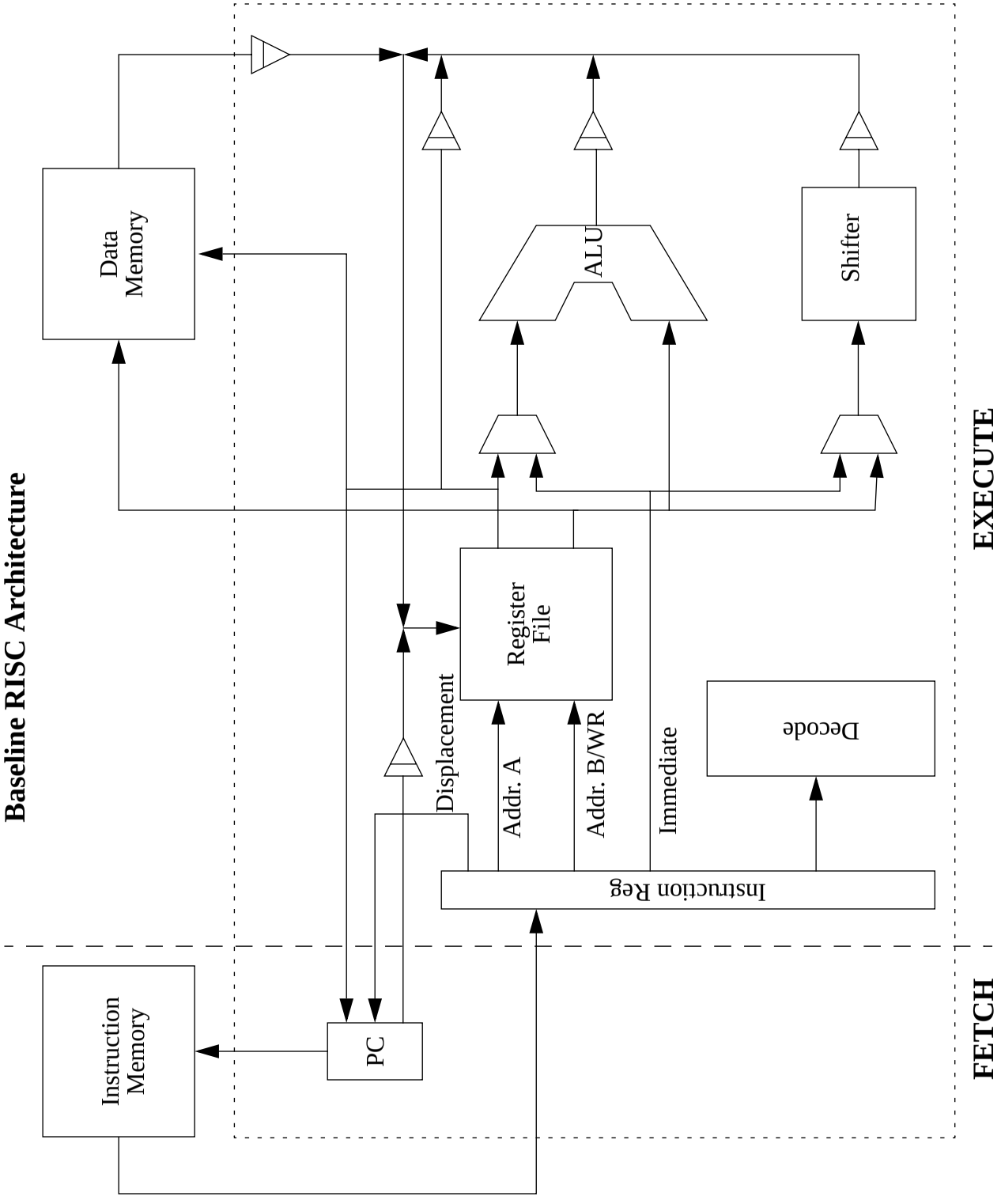
TBIT copies the bit in position *offset* to the F flag of the PSR

WAIT suspends program execution until an interrupt occurs (or for ever, if interrupts are not implemented).

ZRXB converts the 8-bit operand in Rsrc to 16 bits in Rdest with zeros-extension.

Table 1: COND Values for Jcond, Bcond, and Scond

Mnemonic	Bit Pattern	Description	PSR Values
EQ	0 0 0 0	Equal	Z=1
NE	0 0 0 1	Not Equal	Z=0
GE	1 1 0 1	Greater than or Equal	N=1 or Z=1
CS	0 0 1 0	Carry Set	C=1
CC	0 0 1 1	Carry Clear	C=0
HI	0 1 0 0	Higher than	L=1
LS	0 1 0 1	Lower than or Same as	L=0
LO	1 0 1 0	Lower than	L=0 and Z=0
HS	1 0 1 1	Higher than or Same as	L=1 or Z=1
GT	0 1 1 0	Greater Than	N=1
LE	0 1 1 1	Less than or Equal	N=0
FS	1 0 0 0	Flag Set	F=1
FC	1 0 0 1	Flag Clear	F=0
LT	1 1 0 0	Less Than	N=0 and Z=0
UC	1 1 1 0	Unconditional	N/A
	1 1 1 1	Never Jump	N/A


Baseline RISC Architecture
FETCH
EXECUTE
Instruction Memory
PC
Data Memory
ALU
Shifter
Register File
Displacement
Addr. A
Addr. B/WR
Immediate
Decode
Instruction Reg