CMP N 301 Spring 2021



Architecture Project

Objective

To design and implement a simple 5-stage pipelined processor, <u>von Neumann or Harvard</u>. The design should conform to the ISA specification described in the following sections.

Introduction

The processor in this project has a RISC-like instruction set architecture. There are eight 4-byte general purpose registers; R_0 , till R7. Another two general purpose registers, One works as a program counter (PC). And the other, works as a stack pointer (SP); and hence; points to the top of the stack. The initial value of SP is (2^20-2) . The memory address space is 1 MB of 16-bit width and is word addressable. (N.B. word = 2 bytes). You are allowed to make the data bus 32 bits to access two consecutive words.

When an interrupt occurs, the processor finishes the currently fetched instructions (instructions that have already entered the pipeline), then the address of the next instruction (in PC) is saved on top of the stack, and PC is loaded from address 1 of the memory. To return from an interrupt, an RTI instruction loads the PC from the top of stack, and the flow of the program resumes from the instruction after the interrupted instruction. Take care of corner cases like Branching.

ISA Specifications

A) Registers

R[0:7]<31:0> ; Eight 32-bit general purpose registers

PC<31:0>; 32-bit program counter SP<31:0>; 32-bit stack pointer

CCR<2:0> ; condition code register

Z<0>:=CCR<0>; zero flag, change after arithmetic, logical, or shift operations N<0>:=CCR<1>; negative flag, change after arithmetic, logical, or shift operations

C<0>:=CCR<2>; carry flag, change after arithmetic or shift operations.

B) Input-Output

IN.PORT<31:0> ; 32-bit data input port OUT.PORT<31:0> ; 32-bit data output port

INTR.IN<0>; a single, non-maskable interrupt

RESET.IN<0>; reset signal

Rsrc ; 1st operand register

Rdst; 2nd operand register and result register field

Offset; Address offset (16 bit)

Imm ; Immediate Value 5 bits for shifting and 16 bits otherwise

Take Care that Some instructions will Occupy more than one memory location

Mnemonic	Function	Grade
	One Operand	
NOP	$PC \leftarrow PC + 1$	
SETC	C ←1	
CLRC	C ←0	
CLR Rdst	Sets Rdst to zeros	
CLK Kusi	Z ←1	
	NOT value stored in register Rdst	
NOT Rdst	$R[Rdst] \leftarrow 1$'s Complement($R[Rdst]$);	
NOT Kust	If (1's Complement(R[Rdst]) = 0): $Z \leftarrow 1$; else: $Z \leftarrow 0$;	
	If (1's Complement(R[Rdst]) < 0): N \leftarrow 1; else: N \leftarrow 0	
	Increment value stored in Rdst	
DIC D1	$R[Rdst] \leftarrow R[Rdst] + 1;$	
INC Rdst	If $((R[Rdst] + 1) = 0)$: $Z \leftarrow 1$; else: $Z \leftarrow 0$;	5 Marks
	If $((R[Rdst] + 1) < 0)$: N \leftarrow 1; else: N \leftarrow 0	
	Decrement value stored in Rdst	
	$R[Rdst] \leftarrow R[Rdst] - 1;$	
DEC Rdst	If $((R[Rdst]-1)=0)$: $Z \leftarrow 1$; else: $Z \leftarrow 0$;	
	If $((R[Rdst] - 1) < 0)$: $N \leftarrow 1$; else: $N \leftarrow 0$	
	Negate the value stored in register Rdst	
	R[Rdst] \leftarrow 2's Complement(R[Rdst]);	
NEG Rdst	If (2's Complement(R[Rdst]) = 0): $Z \leftarrow 1$; else: $Z \leftarrow 0$;	
	If (2's Complement(R[Rdst]) < 0): $N \leftarrow 1$; else: $N \leftarrow 0$	
OUT Rdst		
	$OUT.PORT \leftarrow R[Rdst]$	
IN Rdst	R[Rdst] ←IN.PORT	
) (OVID D1	Two Operands	
MOV Rsrc, Rdst	Move value from register Rsrc to register Rdst	
	Add the values stored in registers Rsrc, Rdst	
ADD Rsrc, Rdst	and store the result in Rdst and updates carry	
	If the result =0 then $Z \leftarrow 1$; else: $Z \leftarrow 0$; If the result <0 then $N \leftarrow 1$; else: $N \leftarrow 0$	
	Subtract the values stored in registers Rsrc, Rdst	
	and store the result in Rdst and updates carry	5 Marks
SUB Rsrc, Rdst	If the result =0 then $Z \leftarrow 1$; else: $Z \leftarrow 0$;	
	If the result <0 then $N \leftarrow 1$; else: $N \leftarrow 0$	
	AND the values stored in registers Rsrc, Rdst	J WIAI NS
AND Rsrc, Rdst	and store the result in Rdst	
AND RSIC, Rust	If the result =0 then $Z \leftarrow 1$; else: $Z \leftarrow 0$;	
	If the result <0 then $N \leftarrow 1$; else: $N \leftarrow 0$	
OR Rsrc, Rdst	OR the values stored in registers Rsrc, Rdst	
	and store the result in Rdst	
 	If the result =0 then $Z \leftarrow 1$; else: $Z \leftarrow 0$;	
	If the result <0 then $N \leftarrow 1$; else: $N \leftarrow 0$	

IADD Rdst,Imm and store the result =0 then Z ←1; clse: Z −0; lf the result =0 then N ←1; clse: N ←0 SHL Rsrc, Imm Shift left Rsrc by #Imm bits and store result in same register Don't forget to update carry SHR Rsrc, Imm Shift right Rsrc by #Imm bits and store result in same register Don't forget to update carry SHR Rsrc, Imm Shift right Rsrc by #Imm bits and store result in same register Don't forget to update carry RLC Rdst Rotate left Rdst; $C \leftarrow R$ Rdst $ < 3 > 0 > \infty $ C RRC Rdst Rotate right Rdst; $C \leftarrow R$ Rdst $ < 3 > 0 > \infty $ C RRC Rdst Rotate right Rdst; $C \leftarrow R$ Rdst $ < 3 > 0 > \infty $ C RRC Rdst R Rdst $ \leftarrow R$ Rdst $ < 3 > 0 > \infty $ C PUSH Rdst X[SP] ← R Rdst $ < 3 > 0 > \infty $ C POP Rdst X[SP] ← R Rdst $ < 3 > 0 > \infty $ C POP Rdst X[SP] ← R Rdst $ < \infty \infty $ LDM Rdst, Imm Load immediate value (16 bit) to register Rdst R[Rdst] A Marks R[Rdst] ← Imm<15:0> LDD Rdst, offset(Rsrc) Store value that is in register Rsrc + offset to register Rsrc1, offset (Rsrc2) STD Rsrc1, offset(Rsrc2) M[Rsrc2 + offset] Branch and Change of Control Operations Jump if zero If (Z=1): PC ← R[Rdst]; (Z=0) Jump if negative If (N=1): PC ← R[Rdst]; (N=0) Jump if			-
If the result = 0 then Z ← 1; else: Z ← 0; If the result < 0 then N ← 1; else: N ← 0	IADD Rdst,Imm	Add the values stored in registers Rdst to Immediate Value	
SHL Rsrc, Imm			
$SHIR Rsrc, Imm \\ Shift left Rsrc by \#lmm bits and store result in same register \\ \hline \textbf{Don't forget to update carry} \\ Shift right Rsrc by \#lmm bits and store result in same register \\ \hline \textbf{Don't forget to update carry} \\ RLC Rdst \\ Rotate left Rdst; C \leftarrow R[Rdst] < 30.0 > &C \\ RRC Rdst \\ R[Rdst] \leftarrow R[Rdst] < 30.0 > &C \\ RRC Rdst \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 30.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 0.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 0.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 0.0 > &C \\ R[Rdst] \leftarrow C&R[Rdst] \leftarrow C&R[Rdst] < 0.0 > &C \\ R[Rdst] \leftarrow C$			
Shift right Rsrc by #Imm bits and store result in same register	SHI Pere Imm		
RLC Rdst Rotate left Rdst; $C \leftarrow R[\text{Rdst}] < 31>$; $R[\text{Rdst}] \leftarrow R[\text{Rdst}] < 30:0 - \&C$ RRC Rdst Rotate right Rdst; $C \leftarrow R[\text{Rdst}] < 30:0 - \&C$ Rotate right Rdst; $C \leftarrow R[\text{Rdst}] < 0>$; $R[\text{Rdst}] \leftarrow R[\text{Rdst}] < 0$; $R[\text{Rdst}] \leftarrow R[\text{Rdst}] \leftarrow R[\text{Rdst}] $; $R[\text{Rdst}] \leftarrow R[\text{Rdst}] \sim R[\text{Rdst}]$	SHE KSIC, IIIIII		
RLC Rdst Rotate left Rdst; $C \leftarrow R[Rdst] < 31 > 0$; $R[Rdst] \leftarrow R[Rdst] < 31 > 0$; $R[Rdst] \leftarrow R[Rdst] < 30 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 30 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 30 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 30 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 30 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 30 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 31 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < 0 > 0 < 0 < 0 < 0$; $R[Rdst] \leftarrow R[Rdst] < R[Rdst] \leftarrow R[Rdst] $	SHR Rsrc Imm		
RIC Rdst $R[Rdst] \leftarrow R[Rdst] < 0:0:0:\&C$ RRC Rdst $R[Rdst] \leftarrow R[Rdst] < 0:0:$ R[Rdst] $\leftarrow C\&R[Rdst] < 0:0:$ Remory Operations PUSH Rdst $X[SP] \leftarrow R[Rdst] \times SP = 2$ POP Rdst $X[SP] \leftarrow R[Rdst] \times SP = 2$ POP Rdst $X[SP] \leftarrow R[Rdst] \times SP = 2$ Load immediate value (16 bit) to register Rdst R[Rdst] $\leftarrow R[Rdst] \leftarrow R[Rdst] \times R[Rdst] \leftarrow R[Rdst] \times R[Rdst] $	5111(1010, 111111		
$R[Rdst] \leftarrow R[Rdst] < 30:0-2-2-2-2-2-2-2-2-2-2-2-2-2-2-2-2-2-2-$	RLC Rdst	, , ,	
RRC Rdst $ \leftarrow C\&R[Rdst] < 31:1>;$ Memory Operations PUSH Rdst $ \times SP+=2 R[Rdst] \le SP-=2$ POP Rdst $ \times SP+=2 R[Rdst] \le SP-=2$ LDM Rdst, Imm $ \times SP+=2 R[Rdst] \le SP-=2 $ LDM Rdst, Imm $ \times SP+=2 R[Rdst] \le SP-=2 $ LDD Rdst, offset(Rsrc) $ \times SIR = SIR = SIR $ Load value from memory address Rsrc + offset to register Rdst Rdst R[Rdst] \leftarrow M[Rsrc + offset]; STD Rsrc1, offset(Rsrc2) Store value that is in register Rsrc1 to memory location Rsrc2 + offset $ \times SIR = SIR =$			
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	RRC Rdst		
PUSH Rdst $X[SP] \leftarrow R[Rdst]; SP=2$ POP Rdst $SP+=2; R[Rdst] \leftarrow X[SP];$ LDM Rdst, Imm Load immediate value (16 bit) to register Rdst $R[Rdst] \leftarrow Imm < 15:0 >$ LDD Rdst, offset(Rsrc) Load value from memory address Rsrc + offset to register Rdst $R[Rdst] \leftarrow M[Rsrc + offset];$ STD Rsrc1, offset(Rsrc2) Store value that is in register Rsrc1 to memory location $Rsrc2 + offset$ $M[Rsrc2 + offset] \leftarrow R[Rsrc1];$ Branch and Change of Control Operations JZ Rdst Jump if zero $If(Z=1): PC \leftarrow R[Rdst]; (Z=0)$ JN Rdst Jump if negative $If(N=1): PC \leftarrow R[Rdst]; (N=0)$ JC Rdst Jump if negative $If(C=1): PC \leftarrow R[Rdst]; (C=0)$ JMP Rdst Jump $PC \leftarrow R[Rdst]; (C=0)$ JMP Rdst $PC \leftarrow R[Rdst]$ CALL Rdst $(X[SP] \leftarrow PC + 1; sp = 2; PC \leftarrow R[Rdst])$ RET $sp + 2, PC \leftarrow X[SP]$	rice reast		
$POP \ Rdst \qquad SP+=2; R[\ Rdst\] \leftarrow X[SP];$ $LDM \ Rdst, Imm \qquad Load \ immediate \ value \ (16 \ bit) \ to \ register \ Rdst \\ R[\ Rdst\] \leftarrow Imm<15:0>$ $LDD \ Rdst, \\ offset(Rsrc) \qquad Load \ value \ from \ memory \ address \ Rsrc + offset \ to \ register \\ Rdst \\ R[\ Rdst\] \leftarrow M[Rsrc + offset];$ $STD \ Rsrc1, \\ offset(Rsrc2) \qquad Store \ value \ that \ is \ in \ register \ Rsrc1 \ to \ memory \ location \\ Rsrc2 + offset \\ M[Rsrc2 + offset];$ $Branch \ and \ Change \ of \ Control \ Operations$ $JUmp \ if \ zero$ $JUmp \ if \ negative \\ If \ (N=1): \ PC \leftarrow R[\ Rdst\]; \ (Z=0)$ $JN \ Rdst \qquad If \ (N=1): \ PC \leftarrow R[\ Rdst\]; \ (N=0)$ $JUmp \ if \ negative \\ If \ (C=1): \ PC \leftarrow R[\ Rdst\]; \ (C=0)$ $JUmp \ if \ negative \\ If \ (C=1): \ PC \leftarrow R[\ Rdst\]; \ (C=0)$ $JMP \ Rdst \qquad Jump \\ PC \leftarrow R[\ Rdst\]$ $CALL \ Rdst \qquad (X[SP] \leftarrow PC + 1; \ sp-2; \ PC \leftarrow R[\ Rdst\])$ $RET \qquad Sp+=2, \ PC \leftarrow X[SP]$		Memory Operations	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	PUSH Rdst	$X[SP] \leftarrow R[Rdst]; SP=2$	
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	POP Rdst	$SP+=2$; $R[Rdst] \leftarrow X[SP]$;	
$ \begin{array}{c c} R[Rdst] \leftarrow Imm < 15:0> \\ Load value from memory address Rsrc + offset to register \\ Rdst \\ R[Rdst] \leftarrow M[Rsrc + offset]; \\ \hline STD Rsrc1, \\ offset(Rsrc2) \\ \hline Store value that is in register Rsrc1 to memory location \\ Rsrc2 + offset \\ M[Rsrc2 + offset] \leftarrow R[Rsrc1]; \\ \hline \hline Branch and Change of Control Operations \\ JUmp if zero \\ If (Z=1): PC \leftarrow R[Rdst]; (Z=0) \\ Jump if negative \\ If (N=1): PC \leftarrow R[Rdst]; (N=0) \\ Jump if negative \\ If (C=1): PC \leftarrow R[Rdst]; (C=0) \\ Jump Rdst \\ Jump \\ PC \leftarrow R[Rdst] \\ Jump \\ PC \leftarrow R[Rdst] \\ Sump \\ PC \leftarrow R[Rdst] \\ RET \\ \hline Sp+=2, PC \leftarrow X[SP] \\ \hline $	I DM D dat Imm	Load immediate value (16 bit) to register Rdst	
$ \begin{array}{c} LDD \ Rdst, \\ offset(Rsrc) \\ \hline \\ STD \ Rsrc1, \\ offset(Rsrc2) \\ \hline \\ Store \ value \ that \ is \ in \ register \ Rsrc1 \ to \ memory \ location \\ Rsrc2 + offset \\ M[Rsrc2 + offset] \leftarrow R[Rsrc1]; \\ \hline \\ \hline \\ Branch \ and \ Change \ of \ Control \ Operations \\ \hline \\ JZ \ Rdst \\ \hline \\ JZ \ Rdst \\ \hline \\ JUmp \ if \ zero \\ If \ (Z=1): \ PC \leftarrow R[\ Rdst\]; \ (Z=0) \\ Jump \ if \ negative \\ If \ (N=1): \ PC \leftarrow R[\ Rdst\]; \ (N=0) \\ Jump \ if \ negative \\ If \ (C=1): \ PC \leftarrow R[\ Rdst\]; \ (C=0) \\ \hline \\ JMP \ Rdst \\ \hline \\ JMP \ Rdst \\ \hline \\ CALL \ Rdst \\ \hline \\ (X[SP] \leftarrow PC + 1; \ sp-=2; \ PC \leftarrow R[\ Rdst\]) \\ \hline \\ RET \\ \hline \\ Sp+=2, \ PC \leftarrow X[SP] \\ \hline \end{array} $	LDM Rust, Imm	$R[Rdst] \leftarrow Imm < 15:0 >$	
$ \begin{array}{c} Rdst \\ offset(Rsrc) \\ \hline \\ R[Rdst] \leftarrow M[Rsrc + offset]; \\ \hline \\ STD Rsrc1, \\ offset(Rsrc2) \\ \hline \\ \\ Store value that is in register Rsrc1 to memory location \\ Rsrc2 + offset \\ M[Rsrc2 + offset] \leftarrow R[Rsrc1]; \\ \hline \\ \\ Branch \ and \ Change \ of \ Control \ Operations \\ \hline \\ JZ \ Rdst \\ \hline \\ JUmp \ if \ zero \\ If (Z=1): PC \leftarrow R[\ Rdst\]; (Z=0) \\ \hline \\ JN \ Rdst \\ \hline \\ JUmp \ if \ negative \\ If (N=1): PC \leftarrow R[\ Rdst\]; (N=0) \\ \hline \\ JUmp \ if \ negative \\ \hline \\ JUmp \ if \ negative \\ \hline \\ JC \ Rdst \\ \hline \\ JUmp \ if \ negative \\ \hline \\ JUmp \ if \ negative \\ \hline \\ JC \ Rdst \\ \hline \\ JUmp \ if \ negative \\ \hline \\ JUmp \ if \ negative \\ \hline \\ JC \ Rdst \\ \hline \\ JC \ Rdst \\ \hline \\ JC \ Althoration \ Al$	I DD Ddat	Load value from memory address Rsrc + offset to register	4 Marks
STD Rsrc1, offset(Rsrc2) Store value that is in register Rsrc1 to memory location Rsrc2 + offset M[Rsrc2 + offset] \leftarrow R[Rsrc1]; Branch and Change of Control Operations Jump if zero If (Z=1): PC \leftarrow R[Rdst]; (Z=0) Jump if negative If (N=1): PC \leftarrow R[Rdst]; (N=0) Jump if negative If (C=1): PC \leftarrow R[Rdst]; (C=0) Jump if negative If (C=1): PC \leftarrow R[Rdst]; (C=0) Jump Amarks Jump CALL Rdst (X[SP] \leftarrow PC + 1; sp-2; PC \leftarrow R[Rdst]) RET Store value that is in register Rsrc1 to memory location Rsrc2 + offset M[Rsrc2 + offset Rsrc1]; Branch and Change of Control Operations Jump if negative If (N=1): PC \leftarrow R[Rdst]; (N=0) 3 Marks Jump PC \leftarrow R[Rdst] CALL Rdst \Rightarrow PC \leftarrow R[Rdst]	,	Rdst	Tiviairs
$STD RSrc1, offset(Rsrc2) \\ RSrc2 + offset \\ M[Rsrc2 + offset] \leftarrow R[Rsrc1]; \\ \hline \\ Branch \ and \ Change \ of \ Control \ Operations \\ \\ JUmp \ if \ zero \\ If \ (Z=1): \ PC \leftarrow R[\ Rdst\]; \ (Z=0) \\ \\ Jump \ if \ negative \\ If \ (N=1): \ PC \leftarrow R[\ Rdst\]; \ (N=0) \\ \\ Jump \ if \ negative \\ If \ (C=1): \ PC \leftarrow R[\ Rdst\]; \ (C=0) \\ \hline \\ JMP \ Rdst \\ Dump \\ PC \leftarrow R[\ Rdst\] \\ \hline CALL \ Rdst \\ (X[SP] \leftarrow PC + 1; \ sp-2; \ PC \leftarrow R[\ Rdst\]) \\ \hline RET \\ Sp+=2, \ PC \leftarrow X[SP] \\ \hline \\$	onsei(Rsic)	$R[Rdst] \leftarrow M[Rsrc + offset];$	
offset(Rsrc2) Rsrc2 + offset M[Rsrc2 + offset] \leftarrow R[Rsrc1]; Branch and Change of Control Operations Jump if zero If (Z=1): PC \leftarrow R[Rdst]; (Z=0) Jump if negative If (N=1): PC \leftarrow R[Rdst]; (N=0) JC Rdst If (C=1): PC \leftarrow R[Rdst]; (C=0) JMP Rdst PC \leftarrow R[Rdst] CALL Rdst (X[SP] \leftarrow PC + 1; sp-=2; PC \leftarrow R[Rdst]) RET Rsrc2 + offset M[Rsrc2 + offset Ampreciation Jump if negative If (C=0) 3 Marks Sump PC \leftarrow R[Rdst]	STD Parol	Store value that is in register Rsrc1 to memory location	
Branch and Change of Control Operations Jump if zero If $(Z=1)$: $PC \leftarrow R[Rdst]$; $(Z=0)$ Jump if negative If $(N=1)$: $PC \leftarrow R[Rdst]$; $(N=0)$ Jump if negative If $(C=1)$: $PC \leftarrow R[Rdst]$; $(C=0)$ Jump if negative If $(C=1)$: $PC \leftarrow R[Rdst]$; $(C=0)$ JMP Rdst $PC \leftarrow R[Rdst]$ CALL Rdst $(X[SP] \leftarrow PC + 1$; $SP = 2$; $PC \leftarrow R[Rdst]$) RET $SP + = 2$, $PC \leftarrow X[SP]$	· · · · · · · · · · · · · · · · · · ·		
$ \begin{tabular}{lllllllllllllllllllllllllllllllllll$	onset(Rsrc2)	$M[Rsrc2 + offset] \leftarrow R[Rsrc1];$	
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$		Branch and Change of Control Operations	
JN Rdst Jump if negative If (N=1): PC \leftarrow R[Rdst]; (N=0) Jump if negative Jump if negative If (C=1): PC \leftarrow R[Rdst]; (C=0) Jump if negative If (C=1): PC \leftarrow R[Rdst]; (C=0) Jump Jump PC \leftarrow R[Rdst] CALL Rdst (X[SP] \leftarrow PC + 1; sp-=2; PC \leftarrow R[Rdst]) RET sp+=2, PC \leftarrow X[SP]		Jump if zero	
$ \begin{array}{c c} JN \ Rdst & If (N=1): \ PC \leftarrow R[\ Rdst\]; (N=0) \\ \hline JC \ Rdst & Jump \ if \ negative \\ \hline JMP \ Rdst & Jump \\ \hline PC \leftarrow R[\ Rdst\]; (C=0) \\ \hline JMP \ Rdst & PC \leftarrow R[\ Rdst\] \\ \hline CALL \ Rdst & (X[SP] \leftarrow PC + 1; \ sp-=2; \ PC \leftarrow R[\ Rdst\]) \\ \hline RET & sp+=2, \ PC \leftarrow X[SP] \\ \hline \end{array} $	JZ Rast	If $(Z=1)$: PC \leftarrow R[Rdst]; $(Z=0)$	
JC Rdst Jump if negative If (C=1): PC \leftarrow R[Rdst]; (C=0) Jump Jump Jump PC \leftarrow R[Rdst]; (C=0) 3 Marks Jump PC \leftarrow R[Rdst] CALL Rdst (X[SP] \leftarrow PC + 1; sp-=2; PC \leftarrow R[Rdst]) RET sp+=2, PC \leftarrow X[SP]	INI Didat	Jump if negative	
$ \begin{array}{c c} JC \ Rdst & If \ (C=1): \ PC \leftarrow R[\ Rdst \]; \ (C=0) \\ \hline JMP \ Rdst & Jump \\ PC \leftarrow R[\ Rdst \] \\ \hline CALL \ Rdst & (X[SP] \leftarrow PC + 1; \ sp-=2; \ PC \leftarrow R[\ Rdst \]) \\ \hline RET & sp+=2, \ PC \leftarrow X[SP] \\ \hline \end{array} $	JN Rast	If $(N=1)$: PC \leftarrow R[Rdst]; $(N=0)$	
$ \begin{array}{c c} & \text{If } (C=1): PC \leftarrow R[\ Rdst \]; (C=0) \\ \hline Jump \\ PC \leftarrow R[\ Rdst \] \\ \hline CALL \ Rdst & (X[SP] \leftarrow PC+1; \ sp-=2; \ PC \leftarrow R[\ Rdst \]) \\ \hline RET & sp+=2, \ PC \leftarrow X[SP] \\ \hline \end{array} $	JC Rdst	Jump if negative	
$ \begin{array}{c cccc} & & & & & & & & & \\ & & & & & & & & \\ & & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & \\ & & & \\ & & \\ & & & \\ &$		If $(C=1)$: PC \leftarrow R[Rdst]; $(C=0)$	3 Marks
CALL Rdst $(X[SP] \leftarrow PC + 1; sp.=2; PC \leftarrow R[Rdst])$ RET $sp+=2, PC \leftarrow X[SP]$	IMD D.det	Jump	
RET $sp+=2$, $PC \leftarrow X[SP]$	JIVII Kust	PC ←R[Rdst]	
	CALL Rdst	$(X[SP] \leftarrow PC + 1; sp=2; PC \leftarrow R[Rdst])$	
DTI gn+-2: DC / V[SD]: Elega restored	RET	$sp+=2, PC \leftarrow X[SP]$	
$\frac{sp-z, re \leftarrow A[sr], rrags restored}{sp-z}$	RTI	$sp+=2; PC \leftarrow X[SP]; Flags restored$	

	Input Signals	Grade
Reset	$PC \leftarrow M[0]$ //memory location of zero	0.5 Mark
Interrupt	$X[Sp] \leftarrow PC$; $sp=2$; $PC \leftarrow M[1]$; $Flags preserved$	3 Mark

Phase1 Requirement: Report Containing:

- Instruction format of your design
 - Opcode of each instruction
 - Instruction bits details
- Schematic diagram of the processor with data flow details. (Including Branching even if you are not planning to implement it).
 - ALU / Registers / Memory Blocks
 - Dataflow Interconnections between Blocks & its sizes

- Control Unit detailed design
- Pipeline stages design
 - Pipeline registers details (Size, Input, Connection, ...)
 - Pipeline hazards and your solution including
 - i. Data Forwarding
 - ii. Static Branch Prediction

Phase2 Requirement

- Implement and integrate your architecture
 - VHDL Implementation of each component of the processor
 - VHDL file that integrates the different components in a single module
- Simulation Test code that reads a program file and executes it on the processor.
 - Setup the simulation wave
 - Load Memory File & Run the test program
- Assembler code that converts assembly program (Text File) into machine code according to your design (Memory File)
- Report that contains any design changes after phase1
- Report that contains pipeline hazards considered and how your design solves it.

Project Testing

- You will be given different test programs. You are required to compile and load it onto the RAM and **reset** your processor to start executing from memory location 0000h. Each program would test some instructions (you should notify the TA if you haven't implemented or have logical errors concerning some of the instruction set).
- You MUST prepare a waveform using do files with the main signals showing that your processor is working correctly (R0-R7, PC,SP,Flags,CLK,Reset,Interrupt, IN.port,Out.port).

Evaluation Criteria

- Each project will be evaluated according to the number of instructions that are implemented, and Pipelining hazards handled in the design. Table 2 shows the evaluation criteria in detail.
- Failing to implement a working processor will nullify your project grade. <u>No credits will be given to individual modules or a non-working processor.</u>
- Unnecessary latching or very poor understanding of underlying hardware will be penalized.
- Individual Members of the same team can have different grades, you can get a zero grade if you didn't work while the rest of the team can get fullmark, Make sure you balance your Work distribution.

Table 2: Evaluation Criteria

Marks Distribution	Instructions	Stated above (14.5 marks) that will be scaled to 21 marks
	Efficient Handling of Hazard	4 marks
	Branching Instructions with control hazards	Bonus that can replace a quiz grade
Bonus Marks	2-bit dynamic branch prediction with address calculation in fetch	2 mark bonus to credit work grade

Team Members

• Each team shall consist of a maximum of four members

Phase 1 Due Date

- Delivery a softcopy on blackboard.
- Week 9, Sunday 16th of May 2021 at 9 am, The discussion will be during the regular lab session.

Project Due Date

- Delivery a softcopy on blackboard.
- Week 12, Monday 14th of June 2021 at 9 am. The demo will be during the regular lab session.

General Advice

- 1. Compile your design on regular bases (after each modification) so that you can figure out new errors early. Accumulated errors are harder to track.
- 2. Start by finishing a working processor that does all one operands only. Integrating early will help you find a lot of errors. You can then add each type of instructions and integrate them into the working processor.
- 3. Use the engineering sense to back trace the error source.
- 4. As much as you can, don't ignore warnings.
- 5. Read the transcript window messages in Modelsim carefully.
- 6. After each major step, and if you have a working processor, save the design before you modify it (use a versioning tool if you can as git & svn).
- 7. Always save the ram files to easily export and import them.
- 8. Start early and give yourself enough time for testing.
- 9. Integrate your components incrementally (i.e. Integrate the RAM with the Registers, then integrate with them the ALU ...).
- 10. Use coding conventions to know each signal functionality easily.
- 11. Try to simulate your control signals sequence for an instruction (i.e. Add) to know if your timing design is correct.
- 12. There is no problem in changing the design after phase1, but justify your changes.
- 13. Always reset all components at the start of the simulation.
- 14. Don't leave any input signal float "U", set it with 0 or 1.
- 15. Remember that your VHDL code is a HW system (logic gates, Flipflops and wires).
- 16. Use Do files instead of re-forcing all inputs each time.