ECE 436

ADVANCED DIGITAL DESIGN

Jackal Version 3.0

A 16-BIT REDUCED INSTRUCTION SET COMPUTER SPECIFICATION FOR THE ECE436 ADVANCED DIGITAL DESIGN COURSE AT THE UNIVERSITY OF VIRGINIA

Created by:

John Lach

Mark Hanson

Arun Thomas

Jackal Version 3.0

TABLE OF CONTENTS

CONTENTS:

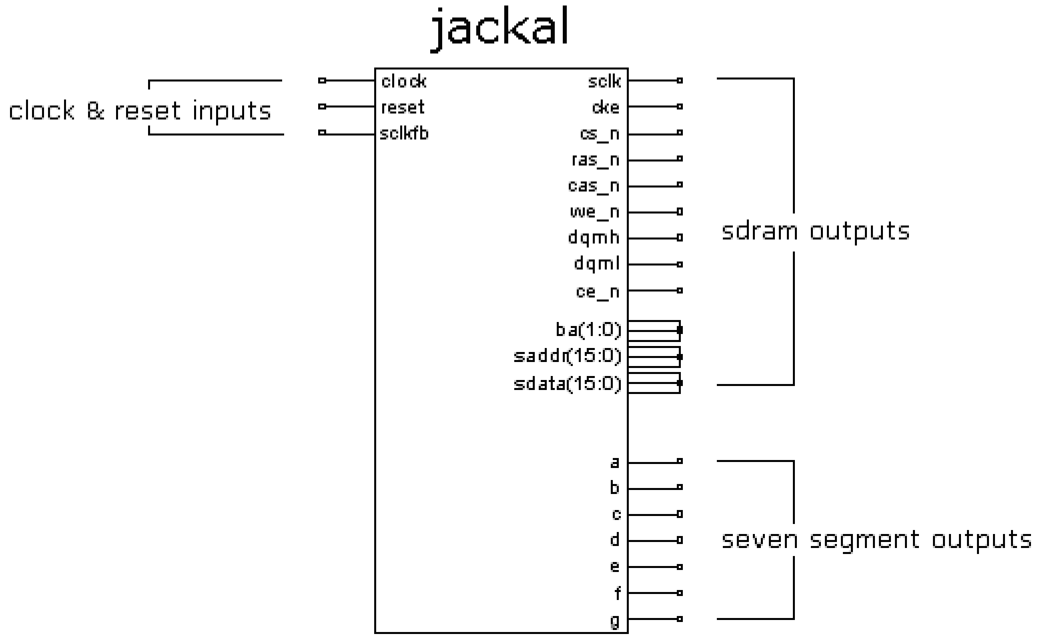
Jackal Version 3.0

INSTRUCTION SET INTRODUCTION

OVERVIEW:
The Jackal is an academic general-purpose reduced instruction set computer (RISC) architecture capable of hosting a wide variety of computational applications. This particular instruction set architecture (ISA) holds to the strictest definition of RISC by providing a very thin core of 16 possible instructions. The ISA's breadth ensures that a student team can design, simulate, synthesize, and place-and-route the processor core to an FPGA prototyping platform in the period of one semester.

A novel aspect of the Jackal ISA lies in its flexibility. While the 16-bit instruction set allocates up to 16 instructions and 16 possible registers, the specification identifies the functionality of 12 of each. This incomplete specification allows students to research and develop a strategy for defining the function and interface to the remaining four instructions and four register operands.

This processor is an educational tool for demonstrating the hardware/software interface while encouraging students to apply skills learned from previous computer engineering courses. As an educational tool that emphasizes simplicity, however, some desirable general-purpose processor (GPP) features have been sacrificed. First, this processor only has a 128KB address space and has limited I/O capabilities. A second deficiency with this processor's architecture is in its lack of interrupt handling capability. Finally, this specification does not include a floating-point unit (FPU); however, this absence does not limit this processor's ability to execute a variety of mathematically-involved algorithms. All students will have to keep cognizant of these deficiencies when writing applications for this processor core. Below, a high-level schematic capture of a previously implemented core depicts the inputs and outputs of the Jackal architecture when mated to an XESS XSA-100 FPGA prototyping platform (see Appendix).



Jackal Version 3.0

INSTRUCTION SET OVERVIEW

SEMANTICS:
The bit numbering scheme will always require that bit strings read from right to left starting with bit 0. The core is little-endian, meaning that bytes at the lowest addresses have the lowest significance. The instruction word size is two bytes or 16-bits. Integer data is represented by two's complement form.

INSTRUCTIONS:
The Jackal uses a 4-bit opcode (allowing for a total of 2^4 instructions) to reference each of the instructions; however, only 12 instructions have been defined. The remaining four instructions may be defined by the student designers. The 12 instructions have been selected to ensure that the core is fully capable of general-purpose processing. Additionally, the core's 12 instructions represent a cross-section of the most frequently used instructions for a general-purpose RISC processor.

REGISTERS:
The Jackal is capable of referencing a total of 2^4 registers with its 4-bit register referencing operands. To permit a greater design space, however, only 12 of the registers operands have been formally assigned to general-purpose registers (R0 to R11). The role of the remaining four undefined operands (U0 to U3) is a task delegated to the student designers.

OPERAND	ROLE
0000	R0
0001	R1
0010	R2
0011	R3

OPERAND	ROLE
0100	R4
0101	R5
0110	R6
0111	R7

OPERAND	ROLE
1000	R8
1001	R9
1010	R10
1011	R11

OPERAND	ROLE
1100	U0
1101	U1
1110	U2
1111	U3

Two addressing modes are available on the Jackal core, register-indirect and immediate. With register-indirect addressing, a register is loaded by using the 16-bits stored in another register to provide the address of data stored at that memory location. With immediate addressing, the low and high bits of a register are loaded through the fixed value in the immediate instruction.

The program counter is a 16-bit register that may be addressable through the 4-bit operand-addressing scheme if the student designers so choose. Resets are triggered logic low and set the data in the PC and registers to 0x0000. Three single-bit condition registers, CRN, CRZ, CRP, store the output of the CMP instruction.

MEMORY:
The Jackal's 16-bit memory address space, is capable of referencing 2^{16} memory locations. Each memory location consists of 2 bytes (16 bits). The memory addresses range from location 0 (0x0000) to 65,535 (0xFFFF). All reads and writes to and from memory are accomplished through the LD/ST instruction. Programs written for the core will start at memory address 0 (0x0000) and may not exceed the range of the address space.

Jackal Version 3.0

INSTRUCTION SET PREVIEW

INSTRUCTION SET CONVENTION:

To minimize the confusion of describing the operation of each instruction, a convention has been established to help with the understanding of this ISA.

CONVENTION	EXAMPLE	DEFINITION
0x	0xFFFF	Denotes Hexadecimal Number FFFF
REG[ ]	REG[SOURCE]	Data in Register Referenced by SOURCE
MEM[ ]	MEM[SOURCE]	Data at Memory Address in SOURCE Register
VAL[ ]	VAL[OFFSET]	Value Denoted by 8-bit OFFSET

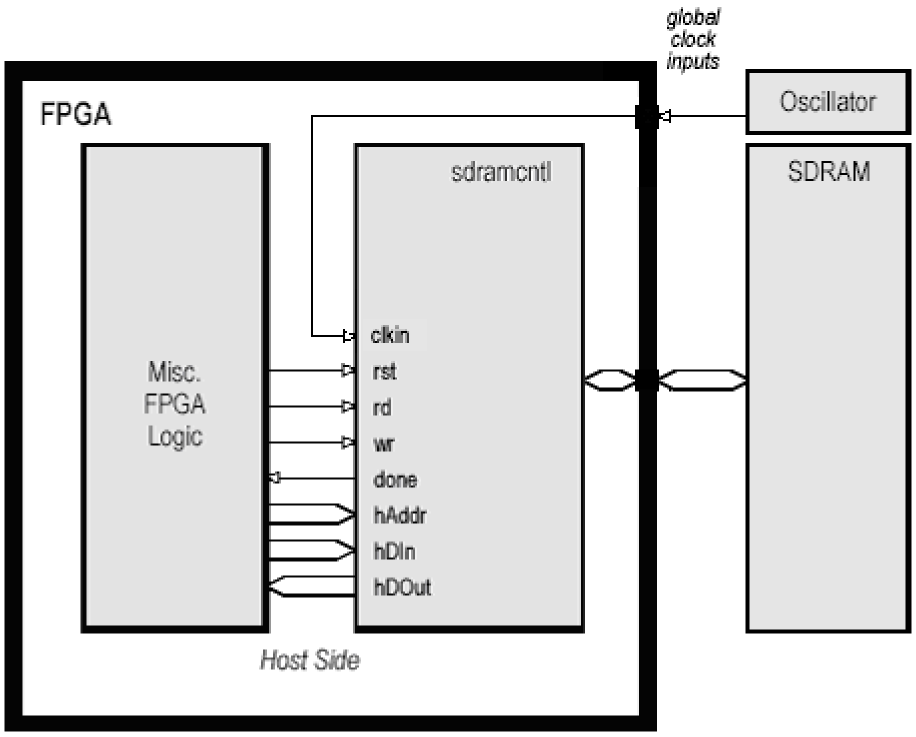
INSTRUCTION SET SUMMARY:

The table below summarizes the instructions explicitly defined by this ISA. To promote simplicity in the ISA, several of the instructions perform a dual purpose. For example, register-to-register data movement can be accomplished with the AND or OR instruction when the SOURCE TWO register contains the data 0xFFFF or 0x0000 respectively. The NAND instruction can act as a logical negation (NOT) when the data at SOURCE TWO is 0xFFFF. The branch functions as a PC-relative jump instruction if the correct MODE applies. Thus, to write efficient assembly routines for this core, student teams will need to explore the nature of each instruction in detail. Further information about each of the instructions can be gained from the latter portion of this document. Please note the presence of undefined instructions (UDI) and undefined operands (UDO). These capabilities may be provided by the students.

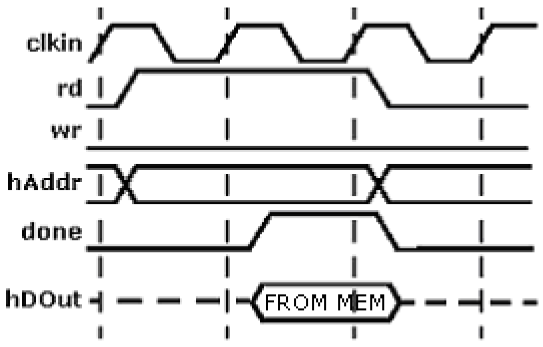
INSTRUCTION	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	0	DESTINATION				SOURCE ONE				SOURCE TWO			
SUB	0	0	0	1	DESTINATION				SOURCE ONE				SOURCE TWO			
AND	0	0	1	0	DESTINATION				SOURCE ONE				SOURCE TWO			
OR	0	0	1	1	DESTINATION				SOURCE ONE				SOURCE TWO			
NAND	0	1	0	0	DESTINATION				SOURCE ONE				SOURCE TWO			
SLA	0	1	0	1	DESTINATION				SOURCE				OFFSET			
SRA	0	1	1	0	DESTINATION				SOURCE				OFFSET			
LD	0	1	1	1	DESTINATION				SOURCE*				MODE			
ST	0	1	1	1	SOURCE				DESTINATION*				MODE			
LIL	1	0	0	0	DESTINATION				IMMEDIATE							
LIH	1	0	0	1	DESTINATION				IMMEDIATE							
CMP	1	0	1	0	UDO				SOURCE ONE				SOURCE TWO			
BRN/BRZ/BRP/JMP	1	0	1	1	MODE				OFFSET							
UDI	1	1	0	0	UDO											
UDI	1	1	0	1	UDO											
UDI	1	1	1	0	UDO											
UDI	1	1	1	1	UDO											
*For the LD/ST instruction, the source/destination operand is a register holding a memory address.																

INSTRUCTION SET MEMORY INTERFACE

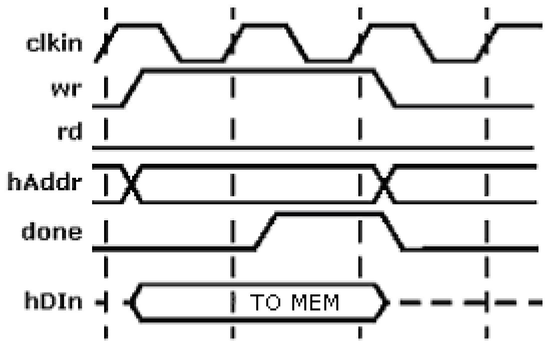
INTERFACE:
The basic black box interface to memory is illustrated below. Note that the SDRAM controller (sdramcntl) is a black box component that cannot be changed, and its VHDL code will be provided to the student design teams. Therefore, only the interface to this subsystem is important.



TIMING:
Below, the clock diagrams for both the memory read and memory write operations illustrate the basic behavior of the sdramcntl. The hAddr, hDIn, and hDOut lines are 16-bit busses. The remaining lines are single-bit standard logic.



memory read operation



memory write operation

ADD

ARITHMETIC INTEGER ADDITION

OPCODE:
0000

OPERATION:
REG[DESTINATION] = REG[SOURCE ONE] + REG[SOURCE TWO]

DESCRIPTION:
The register referenced by source one is added to the register referenced by source two, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by both sources and the destination.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE ONE				SOURCE TWO			
VALUES	0	0	0	0	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
ADD R7 R5 R6   # R7 = R5+R6

BINARY EXAMPLE:

INSTRUCTION	0000	0111	0101	0110
OPERATION	ADD	R7	R5	R6

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE ONE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE TWO	
BBBB	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SUB

ARITHMETIC INTEGER SUBTRACTION

OPCODE:
0001

OPERATION:
REG[DESTINATION] = REG[SOURCE ONE] - REG[SOURCE TWO]

DESCRIPTION:
The register referenced by source two is subtracted from the register referenced by source one, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by both sources and the destination.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE ONE				SOURCE TWO			
VALUES	0	0	0	1	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
SUB R7 R5 R6 # R7 = R5-R6

BINARY EXAMPLE:

INSTRUCTION	0001	0111	0101	0110
OPERATION	SUB	R7	R5	R6

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE ONE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE TWO	
BBBB	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

AND

Logical Bitwise And

Opcode:
0010

Operation:
REG[DESTINATION] = REG[SOURCE ONE] & REG[SOURCE TWO]

Description:
The register referenced by source one is ANDed with the register referenced by source two, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by both sources and the destination.

Instruction Word Format:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE ONE				SOURCE TWO			
VALUES	0	0	1	0	C	C	C	C	A	A	A	A	B	B	B	B

Assembly Code Example:
AND R7 R5 R6   # R7 = R5 AND R6

Binary Example:

INSTRUCTION	0010	0111	0101	0110
OPERATION	AND	R7	R5	R6

Operand Reference:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE ONE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE TWO	
BBBB	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

OR

LOGICAL BITWISE INCLUSIVE OR

OPCODE:
0011

OPERATION:
REG[DESTINATION] = REG[SOURCE ONE] | REG[SOURCE TWO]

DESCRIPTION:
The register referenced by source one is inclusively ORed with the register referenced by source two, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by both sources and the destination.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE ONE				SOURCE TWO			
VALUES	0	0	1	1	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
OR R7 R5 R6   # R7 = R5 OR R6

BINARY EXAMPLE:

INSTRUCTION	0011	0111	0101	0110
OPERATION	OR	R7	R5	R6

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE ONE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE TWO	
BBBB	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

NAND

LOGICAL BITWISE NAND

OPCODE:
0100

OPERATION:
REG[DESTINATION] = NOT(REG[SOURCE ONE] & REG[SOURCE TWO])

DESCRIPTION:
The register referenced by source one is NANDed with the register referenced by source two, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by both sources and the destination.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE ONE				SOURCE TWO			
VALUES	0	1	0	0	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
NAND R7 R5 R6 # R7 = R5 NAND R6

BINARY EXAMPLE:

INSTRUCTION	0100	0111	0101	0110
OPERATION	NAND	R7	R5	R6

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE ONE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE TWO	
BBBB	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SLA

ARITHMETIC SHIFT LEFT

OPCODE:
0101

OPERATION:
REG[DESTINATION] = REG[SOURCE] SLA VAL[OFFSET]

DESCRIPTION:
The register referenced by the source is arithmetically shifted left by the value defined in the offset, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by the source and the destination.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE				OFFSET			
VALUES	0	1	0	1	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
SLA R7 R5 6 # R7 = R5 SLA 6

BINARY EXAMPLE:

INSTRUCTION	0101	0111	0101	0110
OPERATION	SLA	R7	R5	6 Bits

OPERAND REFERENCE:

DESTINATION		SOURCE		OFFSET	
CCCC	REGISTER	AAAA	REGISTER	BBBB	SHIFTED
0000	R0	0000	R0	0000	0 Bits
0001	R1	0001	R1	0001	1 Bits
0010	R2	0010	R2	0010	2 Bits
0011	R3	0011	R3	0011	3 Bits
0100	R4	0100	R4	0100	4 Bits
0101	R5	0101	R5	0101	5 Bits
0110	R6	0110	R6	0110	6 Bits
0111	R7	0111	R7	0111	7 Bits
1000	R8	1000	R8	1000	8 Bits
1001	R9	1001	R9	1001	9 Bits
1010	R10	1010	R10	1010	10 Bits
1011	R11	1011	R11	1011	11 Bits
1100	U0	1100	U0	1100	12 Bits
1101	U1	1101	U1	1101	13 Bits
1110	U2	1110	U2	1110	14 Bits
1111	U3	1111	U3	1111	15 Bits

SRA

ARITHMETIC SHIFT RIGHT

OPCODE:
0110

OPERATION:
REG[DESTINATION] = REG[SOURCE] SRA VAL[OFFSET]

DESCRIPTION:
The register referenced by the source is arithmetically shifted right by the value defined in the offset, and the resulting value is placed in the register referenced by the destination. All 16 registers are addressable by the source and the destination.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE				OFFSET			
VALUES	0	1	1	0	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
SRA R7 R5 6 # R7 = R5 SRA 6

BINARY EXAMPLE:

INSTRUCTION	0110	0111	0101	0110
OPERATION	SRA	R7	R5	6 Bits

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

OFFSET	
BBBB	SHIFTED
0000	0 Bits
0001	1 Bits
0010	2 Bits
0011	3 Bits
0100	4 Bits
0101	5 Bits
0110	6 Bits
0111	7 Bits
1000	8 Bits
1001	9 Bits
1010	10 Bits
1011	11 Bits
1100	12 Bits
1101	13 Bits
1110	14 Bits
1111	15 Bits

LD

Load From Memory

Opcode:
0111

Operation:
REG[DESTINATION] ← MEM[SOURCE]

Description:
The memory location referenced by the source register is copied to the register referenced by the destination operand. The MODE must be set to 0110.

Instruction Word Format:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				SOURCE				MODE			
VALUES	0	1	1	1	C	C	C	C	A	A	A	A	B	B	B	B

Assembly Code Example:
LD R7 R5 # R7 = MEM[R5]

Binary Example:

INSTRUCTION	0111	0111	0101	0110
OPERATION	LD/ST	R7	MEM[R5]	LD

Operand Reference:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE	
AAAA	REGISTER
0000	MEM[R0]
0001	MEM[R1]
0010	MEM[R2]
0011	MEM[R3]
0100	MEM[R4]
0101	MEM[R5]
0110	MEM[R6]
0111	MEM[R7]
1000	MEM[R8]
1001	MEM[R9]
1010	MEM[R10]
1011	MEM[R11]
1100	U0
1101	U1
1110	U2
1111	U3

MODE	
BBBB	FUNCTION
0110	LOAD
1001	STORE
others	NOP

ST

STORE TO MEMORY

OPCODE:
0111

OPERATION:
REG[SOURCE] → MEM[DESTINATION]

DESCRIPTION:
The register value referenced by the source operand is copied to the memory location referenced by the destination register. The MODE must be set to 1001.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				SOURCE				DESTINATION				MODE			
VALUES	0	1	1	1	C	C	C	C	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
ST R7 R5 # MEM[R5] = R7

BINARY EXAMPLE:

INSTRUCTION	0111	0111	0101	1001
OPERATION	LD/ST	R7	MEM[R5]	ST

OPERAND REFERENCE:

SOURCE	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

DESTINATION	
AAAA	REGISTER
0000	MEM[R0]
0001	MEM[R1]
0010	MEM[R2]
0011	MEM[R3]
0100	MEM[R4]
0101	MEM[R5]
0110	MEM[R6]
0111	MEM[R7]
1000	MEM[R8]
1001	MEM[R9]
1010	MEM[R10]
1011	MEM[R11]
1100	U0
1101	U1
1110	U2
1111	U3

MODE	
BBBB	FUNCTION
0110	LD
1001	ST
others	NOP

LIL

IMMEDIATE LOW-BYTE LOAD

OPCODE:
1000

OPERATION:
REG[DESTINATION][LOW] = VAL[IMMEDIATE]

DESCRIPTION:
The register referenced by the destination receives the value defined by the immediate portion of the instruction into its low byte. The contents of the destination register's high byte remain unchanged by this instruction.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				IMMEDIATE							
VALUES	1	0	0	0	C	C	C	C	A	A	A	A	A	A	A	A

ASSEMBLY CODE EXAMPLE:
LIL R7 01010110 # R7[LOW] = 01010110

BINARY EXAMPLE:

INSTRUCTION	1000	0111	01010110
OPERATION	LIL	R7	0x56

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

LIH

IMMEDIATE HIGH-BYTE LOAD

OPCODE:
1001

OPERATION:
REG[DESTINATION][HIGH] = VAL[IMMEDIATE]

DESCRIPTION:
The register referenced by the destination receives the value defined by the immediate portion of the instruction into its high byte. The contents of the destination register's low byte remain unchanged by this instruction.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				DESTINATION				IMMEDIATE							
VALUES	1	0	0	1	C	C	C	C	A	A	A	A	A	A	A	A

ASSEMBLY CODE EXAMPLE:
LIH R7 01010110   # R7[HIGH] = 01010110

BINARY EXAMPLE:

INSTRUCTION	1001	0111	01010110
OPERATION	LIH	R7	0x56

OPERAND REFERENCE:

DESTINATION	
CCCC	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

CMP

LOGICAL REGISTER DATA COMPARISON

OPCODE:
1010

OPERATION:
if (REG[SOURCE ONE] – REG[SOURCE TWO]) < 0 (CRN = 1) else (CRN = 0)
if (REG[SOURCE ONE] – REG[SOURCE TWO]) == 0 (CRZ = 1) else (CRZ = 0)
if (REG[SOURCE ONE] – REG[SOURCE TWO]) > 0 (CRP = 1) else (CRP = 0)

DESCRIPTION:
The register referenced by source two is compared to the register referenced by source one. All of the condition code registers are set for each comparison.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				UNUSED				SOURCE ONE				SOURCE TWO			
VALUES	1	0	1	0	X	X	X	X	A	A	A	A	B	B	B	B

ASSEMBLY CODE EXAMPLE:
CMP R5 R6   # R5 >, =, or < R6?

BINARY EXAMPLE:

INSTRUCTION	1010	XXXX	0101	0110
OPERATION	CMP	N/A	R5	R6

OPERAND REFERENCE:

SOURCE ONE	
AAAA	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

SOURCE TWO	
BBBB	REGISTER
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	U0
1101	U1
1110	U2
1111	U3

BRN / BRZ / BRP / JMP

CONDITIONAL BRANCH OR JUMP

OPCODE:
1011

OPERATION:

```
if (MODE==1001)
            PC = (PC+1+VAL[OFFSET])
else if ([(MODE==0110) AND (CRN==1)] OR
      [(MODE==0111) AND (CRZ==1)] OR
      [(MODE==1000) AND (CRP==1)])
            PC = (PC+1+VAL[OFFSET])
else
            PC = (PC+1)
```

DESCRIPTION:
If the MODE is JMP, the PC is set to the part of the program indicated in the assembly code by LABEL. (Note: the offset to reach LABEL is calculated by the assembler based on the incremented PC.) If the MODE is BRN, BRZ, or BRP, then the branch is conditional on CRN, CRZ, or CRP, respectively.

INSTRUCTION WORD FORMAT:

LOCALITY	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GROUPING	OPCODE				MODE				OFFSET							
VALUES	1	0	1	1	C	C	C	C	A	A	A	A	A	A	A	A

ASSEMBLY CODE EXAMPLE:
BRN LABEL BRZ LABEL BRP LABEL JMP LABEL

BINARY EXAMPLE:

INSTRUCTION	1011	0110	01010110
OPERATION	BR/JMP	BRN	0x56

OPERAND REFERENCE:

MODE	
CCCC	TYPE
0110	BRN
0111	BRZ
1000	BRP
1001	JMP
others	NOP

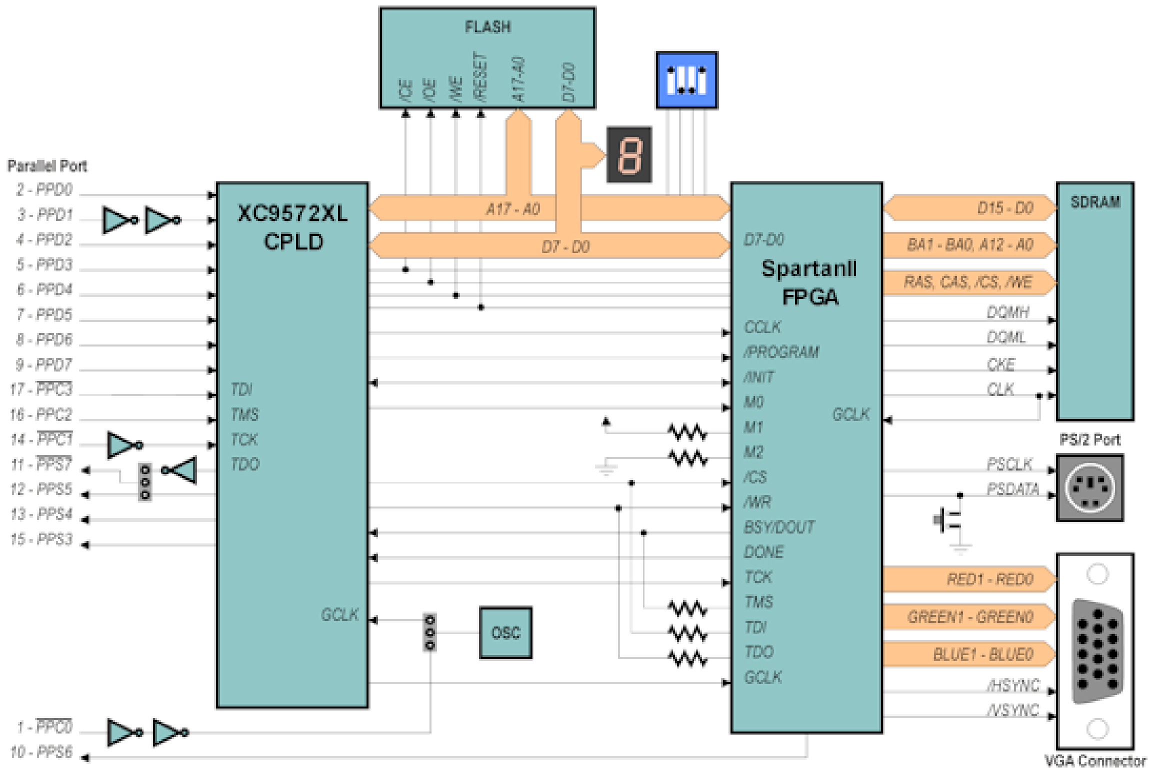
Appendix

Instruction Set Target Hardware

FPGA Prototyping Platform:

The system implementation platform is the fully integrated XESS XSA-100 FPGA prototyping board. This platform is equipped with a state of the art FPGA from the Xilinx Spartan II series (XC2S100). To complement this FPGA are a variety of digitally integrated systems, including a 16 MB SDRAM, a 256 KB Flash, a 100 MHz programmable (divisible by integers) oscillator, an integrated seven-segment display, dip switches, and a parallel port interface. These systems and interfaces will be explored in more detail throughout the semester.

The diagram below, courtesy of the XESS Corporation, illustrates some of the components and interfaces that will be utilized by students implementing this core. The important parts of this diagram are the Spartan II FPGA, the SDRAM, and the seven-segment display. These systems will be directly or indirectly manipulated via student synthesized VHDL code. More information pertaining to the target hardware may be found at: http://www.xess.com/prod026.php3.



FLASH
/CE
/OE
/WE
/RESET
A17-A0
D7-D0
Parallel Port
2 - PPD0
3 - PPD1
4 - PPD2
5 - PPD3
6 - PPD4
7 - PPD5
8 - PPD6
9 - PPD7
17 - PPC3
16 - PPC2
14 - PPC1
11 - PPS7
12 - PPS5
13 - PPS4
15 - PPS3
1 - PPC0
10 - PPS6
XC9572XL
CPLD
TDI
TMS
TCK
TDO
GCLK
A17 - A0
D7 - D0
OSC
SpartanII
FPGA
D7-D0
CCLK
/PROGRAM
/INIT
M0
M1
M2
/CS
/WR
BSY/DOUT
DONE
TCK
TMS
TDI
TDO
GCLK
GCLK
D15 - D0
BA1 - BA0, A12 - A0
RAS, CAS, /CS, /WE
DQMH
DQML
CKE
CLK
SDRAM
PS/2 Port
PSCLK
PSDATA
RED1 - RED0
GREEN1 - GREEN0
BLUE1 - BLUE0
/HSYNC
/VSYNC
VGA Connector