

CS/EE 3710

National Semiconductor CR16
Compact RISC Processor
Baseline ISA and Beyond...

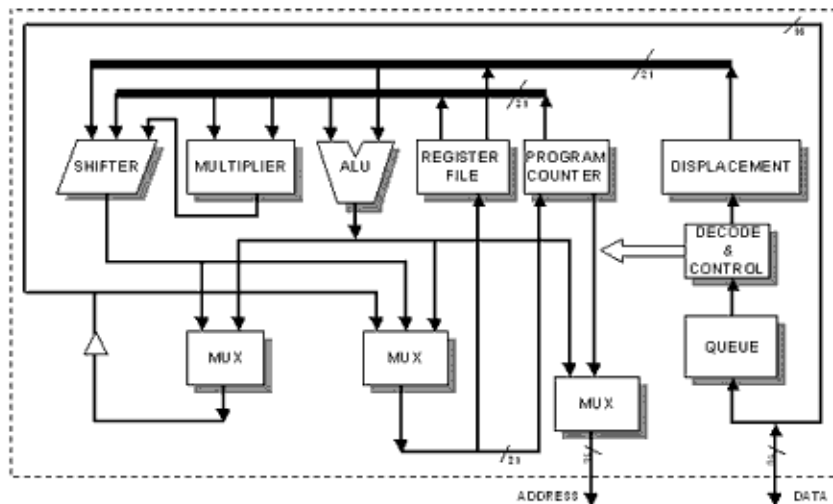
CR16 Architecture

- ◆ Part of a microcontroller family from National Semiconductor
 - 16-bit embedded RISC processor core
 - Available in Synthesizeable Verilog HDL
 - Die size of 0.6 mm² @ 0.25μ
 - 2 Mbytes of linear address space
 - Less than 0.2mA per MHZ @ 3 Volts, 0.35μ
- ◆ This has morphed into the CP3000 family...

CR16 Architecture

- ◆ More specs...
 - Static 0 to 66 MHz clock frequency
 - Atomic memory-direct bit manipulation instructions
 - Save and Restore of Multiple Registers
 - Push and Pop of Multiple Registers
 - Hardware Multiplier Unit for fast 16-bit multiplication

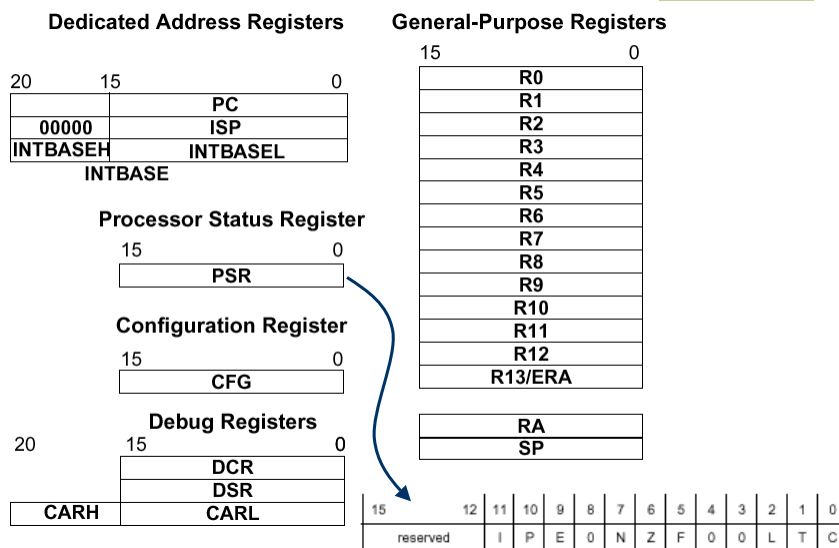
CR16 Block Diagram



CR16 Register Set

- ◆ All registers are 16 bits wide
 - Except address registers which are 21 bits
 - Original version used 18 bits...
- ◆ 16 general purpose registers
- ◆ 8 processor registers
 - 3 dedicated address registers (PC, ISP, INTBASE)
 - 1 Processor Status Register
 - 1 configuration register
 - 3 debug-control registers

CR16 Registers



Processor Registers

- ◆ **PSR** – Processor Status Register
 - C, T, L, F, Z, N, E, P, I bits
 - Carries, conditions, interrupt enables, etc.
- ◆ **INTBASE** - Interrupt Base register
 - Holds the address of the dispatch table for interrupts and traps
- ◆ **ISP** – Interrupt Stack Pointer
 - Points to the lowest address of the last item stored on the interrupt stack

CR16 Instruction Encoding

- ◆ More complex than our version...

15	14	13	12	9	8	5	4	1	0
0	1	i	op code			dest reg		src reg	
1									

Figure B-2. Register to Register Format

15	14	13	12	9	8	5	4	0	
0	0	i	op code			dest reg		imm	

Figure B-3. Short Immediate Value to Register Format

31						16	15	14	13	12	9	8	5	4	0			
imm						0	0	i	op code			dest reg		1	0	0	0	1

Figure B-4. Medium Immediate Value to Register Format

15	14	13	12	9	8	5	4	1	0
op code		i	disp (d ₄ -d ₁)			reg		base reg	
d ₀									

Figure B-8. Load/Store Format, Relative with Short Displacement Value

31						16	15	14	13	12	11	10	9	8	5	4	1	0
disp (d ₁₅ -d ₀)						op code		i	1	0	d ₁₇	d ₁₆	reg		base reg		1	

Figure B-9. Load/Store Format, Relative with Medium Displacement Value

CR16 Instructions

- ◆ Most ALU instructions have two forms
 - **MOV_i** -> **MOVW** or **MOVB**
- ◆ Two-address instruction formal
 - One of the two arguments is also used as destination (Rdest) and is overwritten
 - **ADD R0, R3 => R3 := R0 + R3**
- ◆ Little-Endian data references
 - Least-significant is lowest numbered
 - Both bits and bytes

CR16 Instructions

MOVES

MOV _i	Rsrc/imm, Rdest	Move
MOVX	Rsrc, Rdest	Move with sign extension
MOVZ	Rsrc, Rdest	Move with zero extension
MOVD	imm, (Rdest+1, Rdest)	Move 21-bit immediate to register-pair

INTEGER ARITHMETIC

ADD[U] _i	Rsrc/imm, Rdest	Add
ADDC _i	Rsrc/imm, Rdest	Add with carry
MUL _i	Rsrc/imm, Rdest	Multiply: Rdest(8):= Rdest(8) * Rsrc(8)/Imm Rdest(16):= Rdest(16) * Rsrc(16)/Imm
MULSB	Rsrc, Rdest	Multiply: Rdest(16):= Rdest(8) * Rsrc(8)
MULSW	Rsrc, Rdest	Multiply: (Rdest+1, Rdest):= Rdest(16) * Rsrc(16)
MULUW	Rsrc, Rdest	Multiply: Rsrc = {R0,R1,R8,R9 only} (Rdest+1,Rdest):= Rdest(16) * Rsrc(16);
SUB _i	Rsrc/imm, Rdest	Subtract: (Rdest := Rdest - Rsrc)
SUBC _i	Rsrc/imm, Rdest	Subtract with carry: (Rdest := Rdest - Rsrc)

More CR16 Instructions

INTEGER COMPARISON

CMPi	Rsrc/imm, Rdest	Compare (Rdest – Rsrc)
BEQ0i	Rsrc, disp	Compare Rsrc to 0 and branch if EQUAL Rsrc = (R0,R1,R8,R9 only)
BNE0i	Rsrc, disp	Compare Rsrc to 0 and branch if NOT-EQUAL Rsrc = (R0,R1,R8,R9 only)
BEQ1i	Rsrc, disp	Compare Rsrc to 1 and branch if EQUAL Rsrc = (R0,R1,R8,R9 only)
BNE1i	Rsrc, disp	Compare Rsrc to 1 and branch if NOT-EQUAL Rsrc = (R0,R1,R8,R9 only)

LOGICAL AND BOOLEAN

ANDi	Rsrc/imm, Rdest	Logical AND
ORi	Rsrc/imm, Rdest	Logical OR
Scond	Rdest	Save condition code as boolean
XORi	Rsrc/imm, Rdest	Logical exclusive OR

SHIFTS

ASHUi	Rsrc/imm, Rdest	Arithmetic left/right shift
LSHi	Rsrc/imm, Rdest	Logical left/right shift

Even More CR16 Instructions

BITS

TBIT	Rposition/imm, Rsrc	Test bit in register
SBITi	lposition, 0(Rbase) lposition, disp16(Rbase) lposition, abs	Set a bit in memory; Rbase = (R0, R1, R8, R9)
CBITi	lposition, 0(Rbase) lposition, disp16(Rbase) lposition, abs	Clear a bit in memory Rbase = (R0, R1, R8, R9)
TBITi	lposition, 0(Rbase) lposition, disp16(Rbase) lposition, abs	Test a bit in memory Rbase = (R0, R1, R8, R9)

POPRET	imm, Rdest	Restore registers (similar to POP) and perform JUMP RA or JUMP (RA, ERA), depending on memory model
--------	------------	--

PROCESSOR REGISTER MANIPULATION

LPR	Rsrc, Rproc	Load processor register
SPR	Rproc, Rdest	Store processor register

Still More CR16 Instructions

JUMPS AND LINKAGE

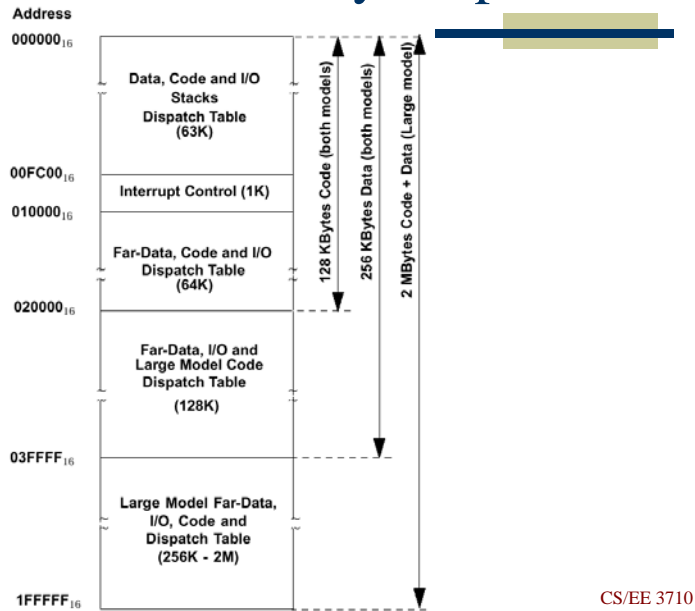
Bcond	disp9 disp17 disp21	Conditional branch using a 9-bit displacement Conditional branch to a small address[S] Conditional branch to a large address[L]
BAL	Rlink, disp17 (Rlink+1, Rlink), disp21	Branch and link to a small address[S] Branch and link to a large address[L]
BR	disp9 disp17 disp21	Branch using a 9-bit displacement Branch to a small address[S] Branch to a large address[L]
EXCP	vector	Trap (vector)
Jcond	Rtarget (Rtarget+1, Rtarget)	Conditional Jump to a small address[S] Conditional Jump to a large address[L]
JAL	Rlink, Rtarget (Rlink+1, Rlink), (Rtarget+1, Rtarget)	Jump and link to a small address[S] Jump and link to a large address[L]
JUMP	Rtarget (Rtarget+1, Rtarget)	Jump to a small address[S] Jump to a large address[L]
RETX		Return from exception
PUSH	imm, Rsrc	Push "imm" number of registers on user stack, starting with Rsrc
POP	imm, Rdest	Restore "imm" number of registers from user stack, starting with Rdest

More and More Instructions

LOAD AND STORE

LOADi	disp(Rbase), Rdest abs, Rdest disp(Rpair+1, Rpair), Rdest	Load (register relative) Load (absolute) Load (far-relative)
STORi	Rsrc, disp(Rbase) Rsrc, disp(Rpair +1, Rpair) Rsrc, abs sm_imm, 0(Rbase) sm_imm, disp(Rbase) sm_imm, abs	Store (register relative) Store (far-relative) Store (absolute) Store small immediate in memory; Rbase = (R0, R1, R8, R9)
LOADM	imm	Load 1 to 4 registers (R2 - R5) from memory, starting at the address in R0, according to imm count value
STORM	imm	Store 1 to 4 registers (R2 - R5) to memory, starting at the address in R1, according to imm count value

CR16 Memory Map



University of Utah

CS/EE 3710

CR16 Exceptions

- ◆ **Interrupt**
 - Exception caused by external activity
 - CR16 recognizes three types, Maskable, Non-maskable, and ISE (In-System Emulator)
- ◆ **Trap**
 - Exception caused by program action
 - Six types: SVC, DVZ, FLG, BPT, TRC, UND
- ◆ **Interrupt process saves PC and PSR on interrupt stack, RETX returns from interrupt**

University of Utah

CS/EE 3710

CR16 Pipeline

- ◆ Three stage pipe
 - Fetch
 - Decode
 - Execute
- ◆ Instruction execution is serialized after an exception
- ◆ Also serialized after LPR, RETX, and EXCP

Our Class Version!

- ◆ Baseline instruction set uses (almost) fixed instruction encoding
- ◆ Detailed description on the web page
 - All instructions are a single 16-bit word
 - All memory references (inst or data) operate on 16-bit words
 - Not all instructions are included
- ◆ Each group will extend the baseline ISA somehow

Baseline ISA

- ◆ ADD, ADDI, SUB, SUBI
- ◆ CMP, CMPI
- ◆ AND, ANDI, OR, ORI, XOR, XORI
- ◆ MOV, MOVI
- ◆ LSH, LSHI (restricted to shift of one)
- ◆ LUI, LOAD, STOR
- ◆ Bcond, Jcond, JAL

Class Encoding

- ◆ In the handout on the web
- ◆ Much more regular than real CR16

Mnemonic	Operands	OP Code 15-12	Rdest 11-8	ImmHi/ OP Code Ext	ImmLo/ Rsrc	Notes (* is Baseline)
				7-4	3-0	
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

Data Types

- ◆ All data is 16-bit
 - **Two's complement** encoding for data
 - **Unsigned** for address manipulation
 - **Boolean** for boolean operations
 - Of course, the ALU doesn't know which is which – they're all 16bit clumps to the ALU!
 - Flags are set for all interpretations
 - The programmer can sort out the flags later

PSR Issues

- ◆ Only **ADD, ADDI, SUB, SUBI, CMP, CMPI** can change the PSR flags
- ◆ **CMP, CMPI** are the same as **SUB, SUBI**
 - But, they affect the PSR differently
- ◆ Only PSR bits **FLCNZ** are needed for baseline implementation
- ◆ **ADD, ADDI, SUB, SUBI** set the **C** on carry out and **F** on overflow
- ◆ **CMP, CMPI** set **Z**, **L** (unsigned), and **N** (signed)

Conditional Jumps/Branches

- ◆ Jumps are absolute
- ◆ Branches are relative to current PC
- ◆ JAL Jump and Link stores the address of the next instruction in Rlink, and jumps to Rtarget
 - Return with JUC Rlink
- ◆ Conditions are derived from PSR bits

Bcond	disp	1100	cond	DispHi	DispLo	* 2s comp displacement
Jcond	Rtarget	0100	cond	1100	Rtarget	*
JAL	Rlink, Rtarget	0100	Rlink	1000	Rtarget	*

University of Utah

CS/EE 3710

Condition Table

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

University of Utah

CS/EE 3710

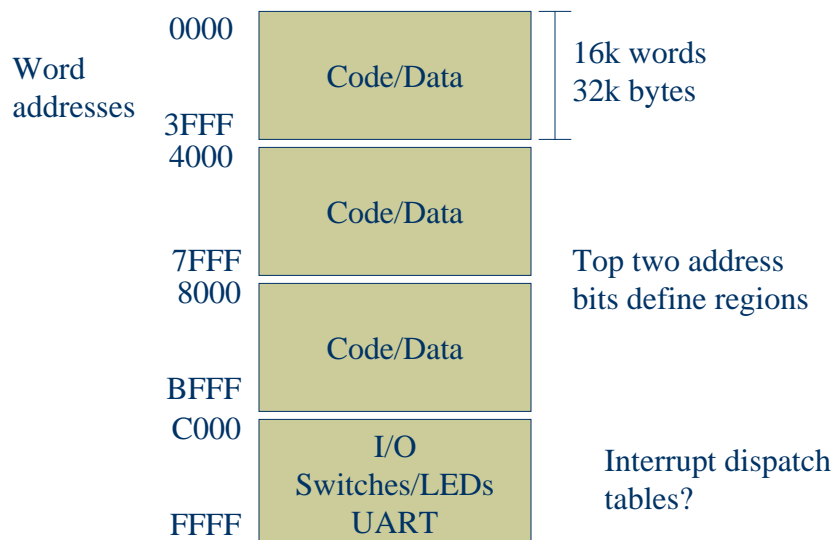
Memory Map

- ◆ 16 bit PC and LOAD/STOR addresses
 - 64k addresses
 - Each address is a 16-bit word
 - So, 128k bytes of data, but organized as words
 - But, only 40k bytes of block RAM on Spartan-3E
 - But, 64M bytes of SDRAM
 - But, SRDAM is a pain...
 - We need to reserve some I/O addresses
 - Up to you, but I recommend using the some top address bits
 - Upper 16k words (32kbytes) as I/O space?

University of Utah

CS/EE 3710

Memory Map



University of Utah

CS/EE 3710