CS/EE 3710

CS/EE 3710

Fall 2019

National Semiconductor CR16

Compact RISC Processor

Baseline ISA and Beyond…

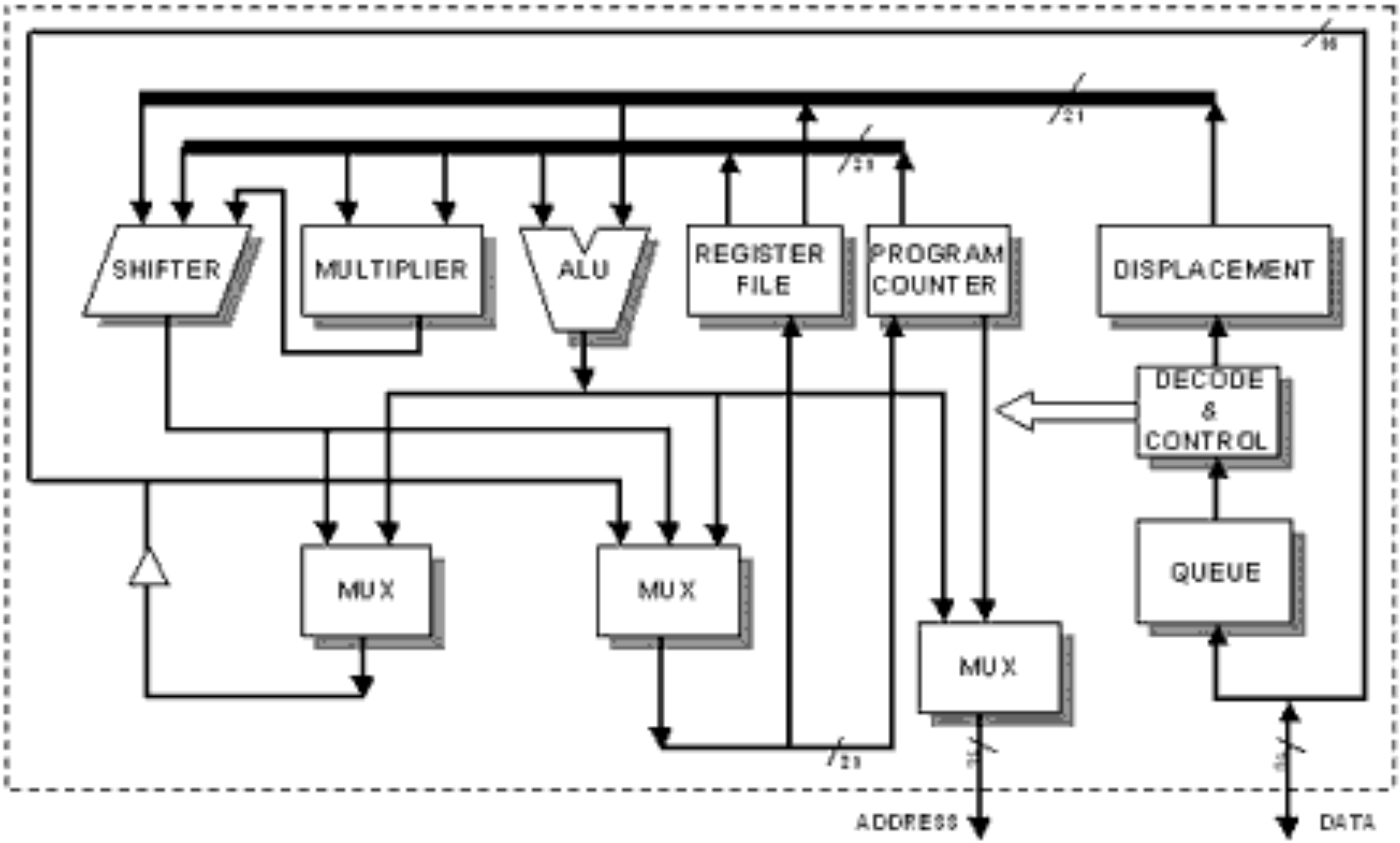
CR16 Architecture

- Part of a microcontroller family from National Semiconductor
  - 16-bit embedded RISC processor core
  - Available in synethesizeable Verilog HDL
  - Die size of 0.6 mm^2 @ 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

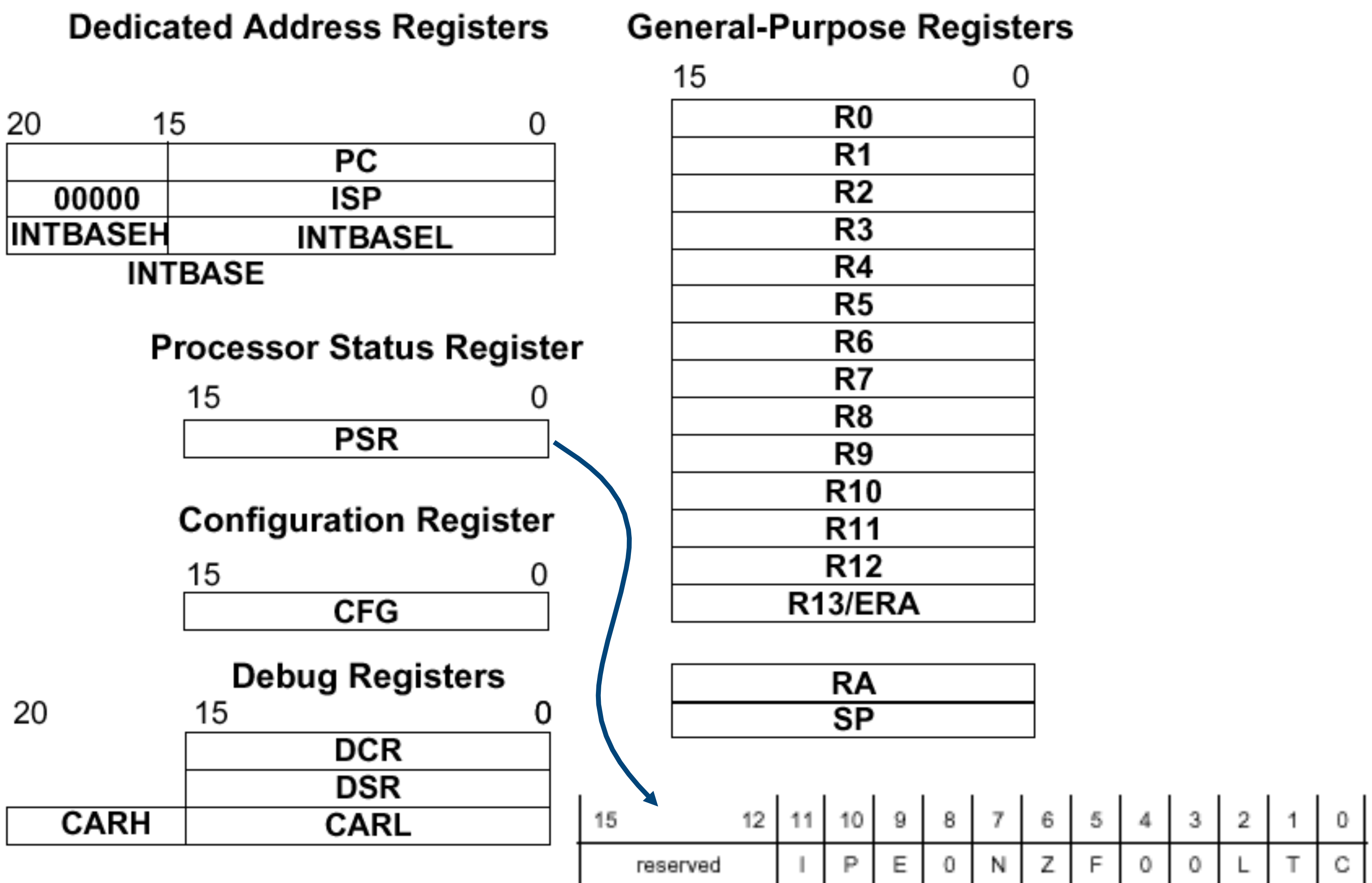
CR16 Block Diagram



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
  - MOVi -> MOVW or MOVB
- Two-address instruction format
  - One of the two arguments is also used as destination (Rdest) and is overwritten
  - ADD R0, R3 => R3 := R3 + R0
- Little-Endian data references
  - Least-significant is lowest numbered
  - Both bits and bytes

CR16 Instructions

MOVES

MOVi	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
ADDCi	Rsrc/imm, Rdest	Add with carry
MULi	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);
SUBi	Rsrc/imm, Rdest	Subtract: (Rdest := Rdest – Rsrc)
SUBCi	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	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Set a bit in memory; Rbase = (R0, R1, R8, R9}
CBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, abs	Clear a bit in memory Rbase = (R0, R1, R8, R9}
TBITi	Iposition, 0(Rbase) Iposition, disp16(Rbase) Iposition, 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

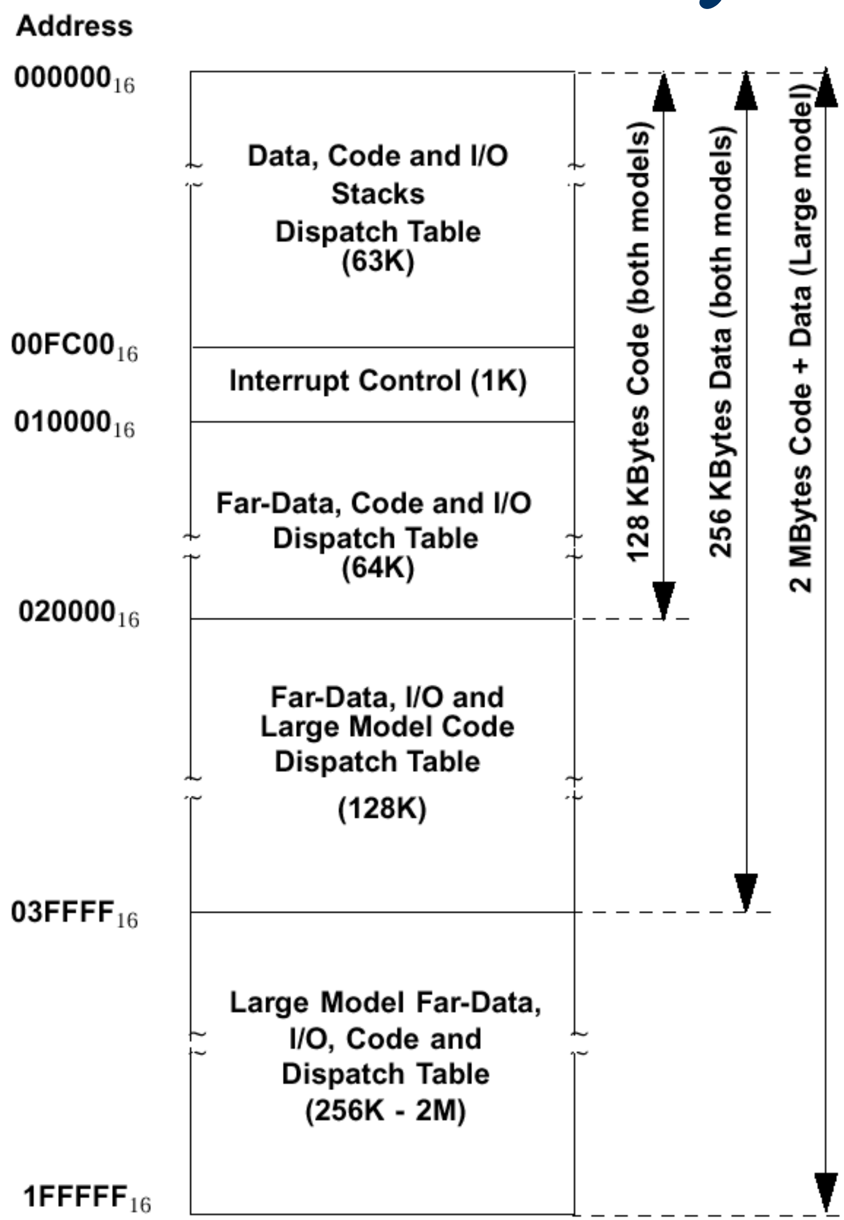
Bcond	disp9 disp17 disp21	Conditional branch using a 9-bit displacement Conditional branch to a small address[S] Conditional branch to a large address[L]
BAL	Rlink, disp17 (Rlink+1, Rlink), disp21	Branch and link to a small address[S] Branch and link to a large address[L]
BR	disp9 disp17 disp21	Branch 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



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

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 CANVAS
  - 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 CANVAS
- Much more regular than real CR16

				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

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 16-bit 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	*

Condition Table

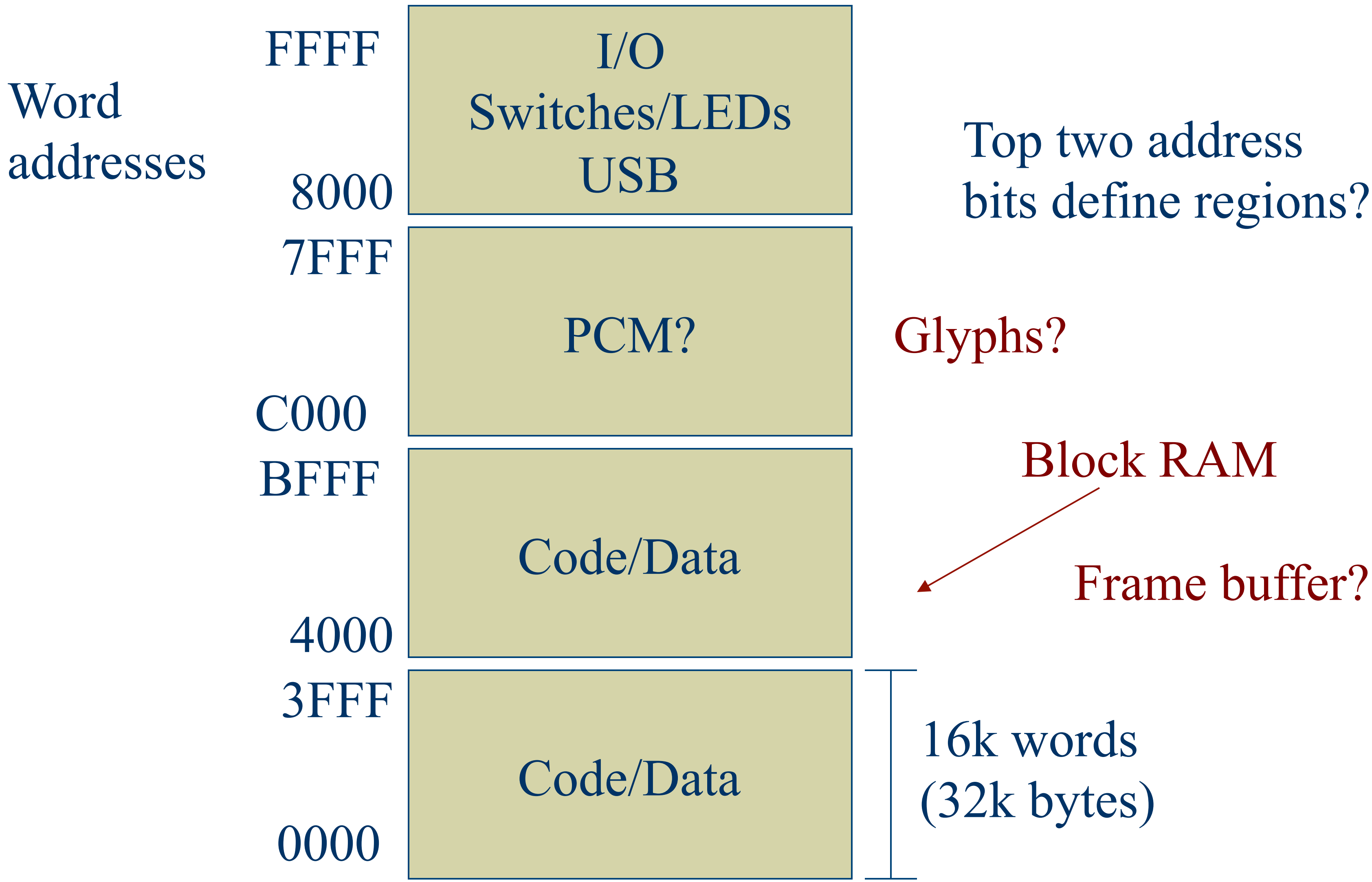
Table 1: COND Values for Jcond, Bcond, and Scond

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

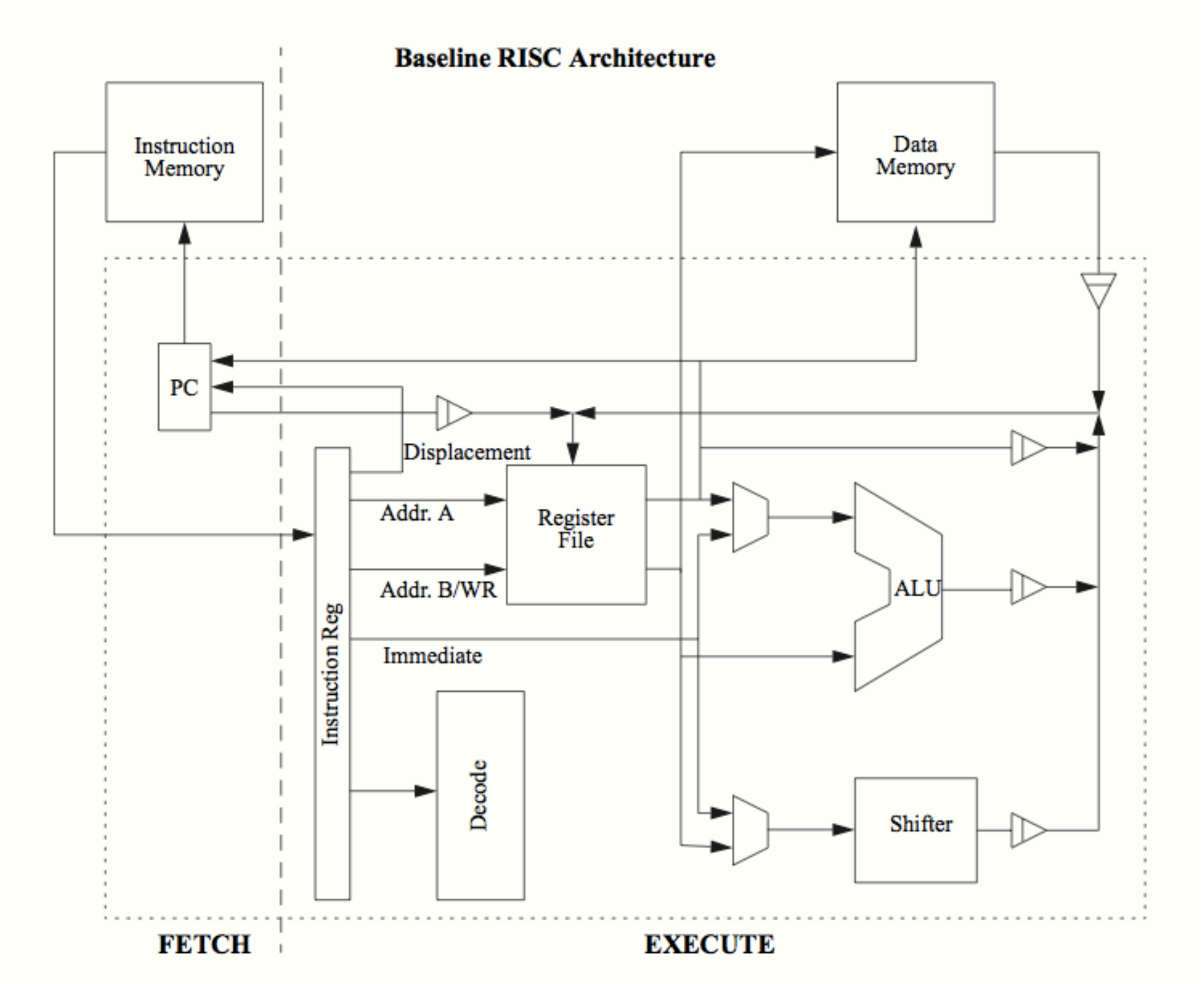
Memory Map

- 16 bit PC and LOAD/STORE addresses
  - 64k addresses
  - Each address is a 16-bit word
  - So, 128k bytes of data, but organized as words
    - But, only 64kbytes of block RAM on Spartan-6
    - But, 16Mbytes of Cellular RAM
    - To use all, must change processor to have 32-bit address and data widths (or at least 24-bit)
  - We need to reserve some I/O addresses
    - Up to you, but I recommend using top address bits
    - Upper 16k words (32kbytes) as I/O space?

Memory Map


Word
addresses
FFFF
I/O
Switches/LEDs
USB
8000
Top two address
bits define regions?
7FFF
PCM?
C000
Glyphs?
Block RAM
BFFF
Code/Data
4000
Frame buffer?
3FFF
Code/Data
0000
16k words
(32k bytes)

Baseline RISC Architecture



Project Tasks

- Design the processor (datapath and FSM).
- Write an assembler software program in your favorite programming language.
- Interface with I/O devices.
- Write the application program in your modified CR-16 assembly code.

Project Checkpoints

1. Propose team name, project customizations and application.
2. Document your register file and ALU.
3. Document your complete datapath including a connection to Block RAM, some memory mapped I/O, and an overall plan for memory.
4. Demonstrate your instruction decoding and how it interacts with your datapath.
5. Document your control FSM that is controlling everything, and demonstrate code running on your processor.
6. Document your I/O system and how it works with your processor. VGA, USB, keyboard, mouse, audio, analog/digital, etc. are all possibilities.

Project Checkpoints

1. Propose team DUE SEPTEMBER 23rd ns and application.
2. Document you DUE SEPTEMBER 30th
3. Document you DUE OCTOBER 14th luding a connection to Block RAM, some memory mapped I/O, and an overall plan for memory.
4. Demonstrate y DUE OCTOBER 21st g and how it interacts with your datapath.
5. Document you DUE OCTOBER 28th ontrolling everything, and demonstrate code running on your processor.
6. Document you DUE NOVEMBER 4th works with your processor. VGA, USB, keyboard, mouse, audio, analog/ digital, etc. are all possibilities.

Checkpoint 0 (due Sept. 23rd)

1. Team name
2. Application
3. Proposed modifications to the ISA
4. I/O peripherals
5. Plan for assembler or other software support
6. Team responsibilities:
   - Each person must be responsible for a h/w (processor or I/O) and a s/w (application or assembler) task.
   - Each task should have a primary and a secondary person responsible for its completion.

Project Meetings

- Project meetings start September 24th.
- Monday at Midnight, project checkpoint documentation is due.
- Demonstration of checkpoint performed during joint meeting with TA and instructor.
- Weekly TA meeting is focused on discussing the next checkpoint.
- At all meetings, all team members are expected to be present and able to answer questions about all parts of the design.

Presentations and Final Report

- All groups present progress to class on Nov. 26th.
- At each checkpoint, groups should prepare thorough written documentation.
- Demo will be in parallel with 4710 demos on December 6th.
- Each group should prepare a poster describing their project for the demo.
- Final report will be due by the end of finals week.