CS31001 COMPUTER ORGANIZATION AND ARCHITECTURE

Debdeep Mukhopadhyay,
CSE, IIT Kharagpur

The Instruction Set Architecture (ISA): From CISC to RISC

Complex Instructions

- The ISA for MIPS was simple.
- Instead, we can have more complex instructions.
  - Complex Instruction Set Computers (CISC)
  - Motivation for designing CISC ISA is to provide an ISA that supports the complex operations and data structures of high level languages (HLLs).

Reasons for the CISC style

- Lesser memory requirements: Memory technology was slow and also small. Complex instructions leads to programs which has lesser memory requirements.
- Hardware based computations are faster than doing it in software:
  - time is saved in fetching and decoding several instructions.

Examples

- VAX-11/780 was an ultimate CISC processor in 1978, supported 22 addressing modes, and has a variable instruction size ranging from 2 to 57.
- For example it has an autoincrement addressing mode:
  - a single instruction can read data from memory, add to the content of a register, store it back in memory and increment the memory pointer.

    (R2)=(R2)+R1, R2=R2+1

  The Complex Instruction can be written as a group of simple instructions (like we have seen in MIPS ISA):

    R4=(R2) #load instruction, I-type

    R4=R4+R1 #add instruction, R-type

    (R2)=R4 #Store instruction, R-type

    R2=R2+1
  - The CISC instruction may be faster than the 4 RISC instructions. But there are other costs, which needs to be investigated.

CISC vs RISC

	CISC		**RISC**
Characteristics	**VAX 11/780**	**Intel 486**	**MIPS R4000**
Number of Instructions	**303**	**235**	**94**
Addressing Modes	**22**	**11**	**1**
Instruction Size (bytes)	**2-57**	**1-12**	**4**
Number of general purpose registers	**16**	**8**	**32**

The Problems of CISC: Why RISC?

- Limited Usage of the Complex Instructions: Complex instructions were used much lesser than expected.
  - Like, sortup reg1, reg2 is a complex instruction to arrange the array from address stored in reg1 to reg2 in a non-descending order is used quite less. Further, the corresponding way of doing the sort hardwired may not be the best way of doing so (the best sorting depends on the type of data).

The Problems of CISC: Why RISC?

- Difficulty of Compilers: With the development of compilers, it was observed that compilers tend to synthesize codes that have simple instructions.
  - Simple ISA also helps in writing efficient compilers.

The Problems of CISC: Why RISC?

- Few Data Types: CISC ISA tends to have a variety of data structures, from simple data types to complex data structures.
- Beneficial to design a system that supports simple data types efficiently, and from which the complex data types can be synthesized.

The Problems of CISC: Why RISC?

- Simple Addressing Modes: CISC designs provide large number of addressing modes.
  - support complex data structures
  - provide flexibility to access operands
- Inefficient Instruction Decoding:
  - Problems arise because of variable instruction execution times, as the time depends on the location of the operands.
  - Variable instruction length

Large Register Sets

- Several researchers have studied the procedure calls in HLLs.
- Study shows that C programs have 12 to 15% call/return instructions.
- Of machine language instructions, it is around 30%.
- Call/return instructions have around 50% memory references: memory is used for local variables, parameters, storing activation record.
- Thus it is important to have proper support for call/return instructions.

Large Register Sets

- Research shows that 1.25 % of the called procedure has more than six arguments.
- More than 93% of them have less than six local variables.
- These figures show that activation records are not large.
  - If we have large number of registers one can avoid memory references for most procedure calls.

Missing Addressing Modes in MIPS

- Index Addressing: Based addressing that we have seen in the MIPS ISA, has a register storing the base address, and an immediate value which is the offset.
- The reverse is also possible, that is the base is specified by an immediate value (and is constant), while the variable index is the content of a register: the register is called index register.
- When the immediate part is 0, the computed address is the same as the value in the base register.
  - This is also called as register index addressing mode: memory is indirectly obtained from a register.
- Some early computers had more than one index registers:
  - the content of one of the index registers is added to the base address, which is either directly specified or indirectly specified.

Indirect Addressing

- Two stage process for obtaining an operand.
- The destination location (either a register or a memory word) is seen.
- It provides an address which is then accessed to obtain the operand.
- If the first stage is a register: register indirect addressing.
- Can be used for executing case statements with jump tables.

Summary of addressing modes

- Implied
- Immediate
- Register
- Base
- PC-relative
- Pseudo-indirect
- Index
- Indirect
  - Often a complex instruction can be expressed through simple addressing modes.

RISC vs CISC

- An ISA has two types of instructions; simple (S) and complex (C) types, both of which takes similar time in a reference CISC implementation.
- Program profiling shows that 95% of the program consist of S type instructions, while the remaining are C type instructions.
- The company is planning to start a RISC version of the processor:
  - The S type instructions will be hardware supported and will have a 20% speed up.
  - The C type instructions will be translated to S-type instructions and will need 3 times for time.
- Which style (CISC or RISC) will give a faster execution of the program?

RISC vs CISC

- CISC: Let an instruction take t time units.
- RISC: Time = 0.95x0.8t + 0.05xt/3
- Speed up of RISC

  = 1/(0.95x0.8+0.05/3)

  =1.1

  Further, the RISC processor is easier to design and is test, and have a shorter time to market.

Combination of RISC and CISC

- Modern processors, like the Intel Pentium tries to combine the advantages in a single machine.
- Front end hardware translators are used that replaces CISC instructions with a sequence of one or more RISC instructions.

A Single Instruction Processor

- How much can we simplify?
- "*Make things* as *simple* as possible, *but not simpler*." – Albert Einstein
- What is the ultimate RISC processor?
  - URISC
- If we neglect the input/outpu, interrupts, scheduling, and other such operations, for computations a single instruction is sufficient!
- Thus extra instructions are added, only for performance.

What is the instruction?

- The instruction does not need any opcode.
- Capability to perform arithmetic is needed:
  - so, we have two operands
  - we re-use the second operand as the destination
  - since, there is no load/store instructions, two memory addresses are needed:
    - source1 and source2/destination
- Further, we need conditional branches:
  - thus the instruction also has a third address, which specifies the target address.
  - The branch condition is specified, by using subtraction as the arithmetic operation, and using the sign of the result as the condition.

The URISC instruction

subtract operand1 from operand2, replace operand2 with the result, and jump to target address in case of negative result

label: urisc dest, src1, target

Note, that the opcode is not needed.

Exiting the program

- The convention will be that the program execution starts at the memory location 1.
- Branching to memory location 0, terminates the program.
- We use the assembler directive, .word to name and initialize one word of memory to be used as an operand
  - stop: .word 0

URISC program for move

```
stop: .word 0
start: urisc dest,dest,+1 #dest = 0
       urisc src,temp,+1 #temp=-(src)
       urisc temp,dest,+1 #dest=-(temp)
       …
```

Assignments

- ex1: uadd dest,src1,src2
- ex2: uswap src1,src2
- ex3: uj label
- ubge: src1, src2, label #if (src1)≥(src2), goto label
- ubeq: src1,src2,label #if (src1)=(src2), goto label

Use at1 and at2 as temporary registers for the assembler.

uadd

urisc at1, at1, +1 #at1 = 0

urisc src1,at1,+1 #at1 = -(src1)

urisc src2,at1,+1 #at1 = -(src1)-(src2)

urisc dest,dest,+1 #dest=0

urisc at1,dest,+1 #dest = -(at1)=(src1)+(src2)

URISC

- URISC stands for Ultra-RISC
- One instruction only !!!!!!!
- No Opcode ……No need for it.. Interesting ???
- Gets better, can be implemented with minimal hardware too

Basic Operations

- ❖Instruction should execute
  - ➢ Subtract
  - ➢ Branch if less than equal
  - ➢ Memory Operations needed

❖Theorem states that any instruction that has this capability can be used as an URISC instruction

Reference: The Ultimate Reduced Instruction Set Computer
Int.J.Elect.Enging Educ., Vol 25, pp 327-334

Motivation

- ❖ How much can RISC be reduced to ?
- ❖ Main idea was to a create a fast, simple computer
- ❖ Simple Instructions means a simple hardware and a faster clock
- ❖ Eliminates the decode stages in other computers
- ❖ URISC is extreme in simplicity

URISC is Turing Compatible

❖Being equivalent to a universal Turing machine essentially means being able to perform any computational task that takes finite input and returns finite output in finitely-many steps.

❖By creating other instructions based on subtract, branch if negative or equal it can be shown that URISC is Turing compatible.

URISC Instruction

- ❖ $b \leftarrow b - a$ (a,b are Registers)
- ❖ If b <=0 then
  PC = Branch Target Address

  else

  PC= PC + 1

❖Branch Addressing can be made relative or implicit

1st Operand = a	2nd Operand = b	Branch Target Address

Machine Level Interpretation

- <L> : <A>, <B>, <P>

L: Instruction Label

P: Jump Target Label

A,B: References to the memory where the operands are located.

Example Programs

- SUB(A,B) // B $\leftarrow$ B - A, no branch

```
SUB:A,B,END
END: …
```

- SETZERO(A) // A $\leftarrow$ 0

```
SETZERO:A,A,END
END: ...
```

- ADD(A,B) // B $\leftarrow$ A + B, Assumes T0 is set to 0

```
ADD0:A,T0,ADD1
ADD1:T0,B,END
END: ...
```

Example Programs

- MOVE(A,B) // B ← A Assumes T0 is set 0

```
MOV0: B,B, MOV1
MOV1: A,T0, MOV2
MOV2: T0,B, END
END: ...
```

- JUMP(Target) // Unconditional Jump

```
JUMP:T0,T0,Target
```

Example Programs

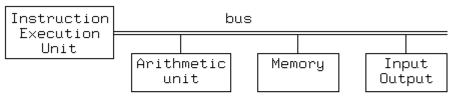
- BEQ(A,Target) //Assumes T0 == 0

```
// IF A == 0 Then PC = Target
BEQ0:A,T0,BEQ2
BEQ1:T0,T0, END
BEQ2:T0,T0, BEQ3
BEQ3:T0,A, Target
END: ...
```

URISC is a Multicycle Processor

- ❖Takes 4 clock cycles to complete a single instruction.
- ❖During each clock cycle a different set of control signals is output from the control unit. These control signals effect the flow of data in the processor/datapath.
- ❖A counter counts what microinstruction we are on. For each value of the counter a different set of microinstructions is output.
- ❖At the end of the instruction (after microinstruction 4 or earlier) the counter is reset to 0, and execution of the next instruction begins

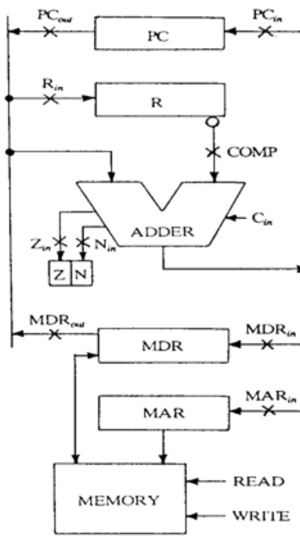
Hardware Implementation



A minimal version of the implementation of the Architecture

❖URISC hardware can be implemented in various forms best suited for optimizing one instruction

Hardware Implementation

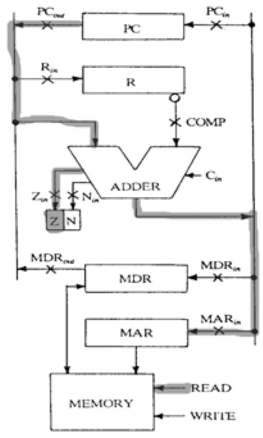


Hardware Implementation

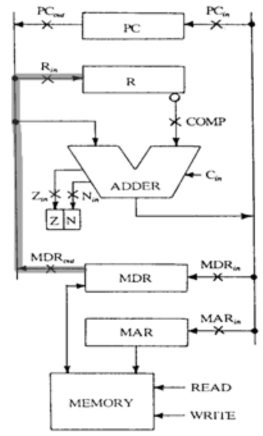
- The URISC Computer uses minimal hardware.
- To implement the instruction we need to:

check to see if PC = 0
load the first operand
increment PC
load the second operand
subtract the operand
store result in to the second operand
increment PC
load target
increment PC
if result is negative, set PC to target

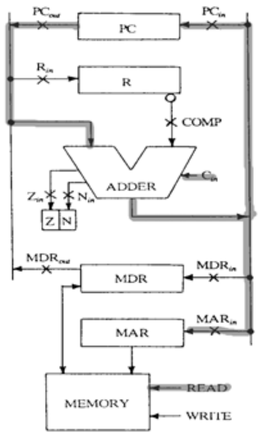
Cycle 1 – PC check to 0



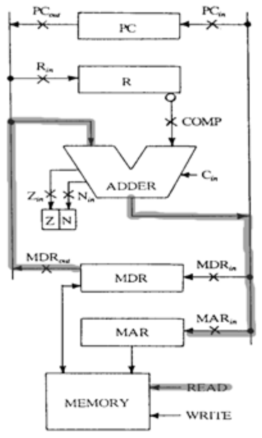
Cycle 2 Loading New Operand



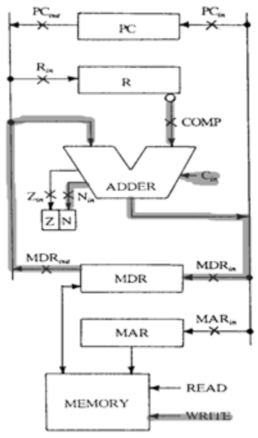
Cycle 3 PC Increment



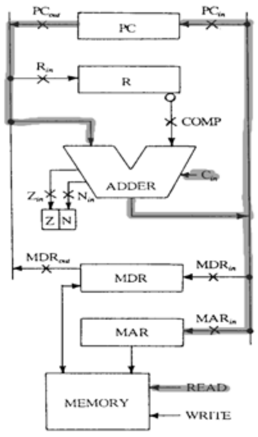
Cycle 4 Load Second Operand



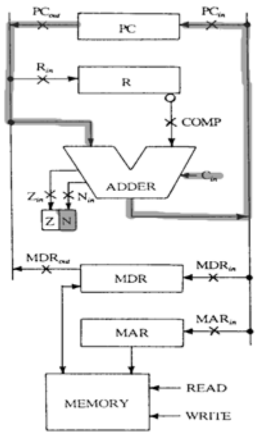
Cycle 5 Substract and Store



Cycle 6 Load Target



Cycle 7 Load target IF PC -ive



Problems with URISC

❖URISC architecture is not competitive

❖The cycles taken by URISC or the execution time per instruction is on an average 75% more than a MIPS multi-cycle architecture

❖But that doesn't prove it to be sub-optimal

Optimal Architectures

- ❖Class of optimal architectures can be thought of as a surface in a multidimensional computer design space
- ❖Taking typical axes of the space to be processor complexity the program size for some benchmark, and the memory traffic required to execute that benchmark, it's clear that URISC fares worse than any other architecture

Optimal Architectures

- ❖The minimal ultimate RISC can only be proven to be suboptimal if a processor can be found that is better when measured along at least one axis of the design space while being no worse along any other axes.

Work on URISC

❖Steve Loughran formally defined, designed and built a 32-bit variant of this architecture as his final-year project at Edinburgh University in 1989

❖Adam Donlin has proposed using an Ultimate RISC as a host for a dynamically reconfigurable FPGA coprocessor in "Self Modifying Circuitry -- A Platform for Trackable Virtual Circuitry" in *Proceedings of FPL the 9th International Workshop, FPL99*, Springer-Verlag, ISSN 0302-9743, Aug 1999.

Work on URISC

❖Paul Frenger wrote published a paper in *ACM Sigplan Notices 35*, 2 (Feb 2000) entitled "The Ultimate RISC: A Zero-Instruction Computer";

❖*ACM Computer Architecture News, 16*, 3 (June 1988), pages 48-55.

❖Univ. of Waterloo URISC: F. Mavaddat and B. Parhami, URISC: The Ultimate Reduced Instruction Set Computer