Instruction Set Comparisons
CS 740

Sept. 24, 1996

Topics
- **Code Examples**
  - Procedure Linkage
  - Sparse Matrix Code
- **Instruction Sets**
  - MIPS
  - PowerPC
  - VAX

Procedure Examples

Procedure linkage
- **Passing of control and data**
- **stack management**

Control
- **Register tests**
- **Condition codes**
- **loop counters**

Code Example

```
int rfact(int n)
{
  if (n <= 1)
    return 1;
  return n*rfact(n-1);
}

int ifact(int n)
{
  int i;
  int result = 1;
  for (i = 2; i <= n; i++)
    result *= i;
  return result;
}
```


MIPS rfact

Registers
- **$4 Argument n**
- **$16 Saved n**
  - Callee save

Stack Frame
- **24 bytes**
- **return PC**
- **$16**

```
rfact:
        subu    $sp,$sp,24       # Allocate stack frame
        sw      $31,20($sp)      # Store return PC
        sw      $16,16($sp)      # Store $16
        move    $16,$4           # $16 <- n
        slt     $2,$16,2         # $2 <- n < 2
        BNE     $2,$0,$L3        # if < goto $L3
        li      $2,0x00000001    # with return value 1
        JAL     rfact            # Call recursively
        subu    $4,$16,1         # with argument n-1
        mult    $16,$2
        mflo    $2               # compute return value
$L3:
        lw      $31,20($sp)      # Restore return PC
        lw      $16,16($sp)      # Restore $16
        addu    $sp,$sp,24       # Remove stack frame
        j       $31              # return
        .end    rfact
```


MIPS ifact

Registers
- **$3 i**
- **$4 n**
- **$5 result**

Stack Frame
- **none**

```
ifact:
#       $3      i
#       $4      n
#       $5      result
        li      $5,0x00000001    # result <- 1
        slt     $2,$4,2          # $2 <- n < 2
        BNE     $2,$0,$L6        # if < goto $L6
        li      $3,0x00000002    # with i <- 2
        mult    $5,$3
$L9:
        mflo    $5               # result *= i
        addu    $3,$3,1          # i++
        slt     $2,$4,$3         # $2 <- n < i
        BEQ     $2,$0,$L9        # if !< goto $L9
        mult    $5,$3            # starting mult
$L6:
        J       $31              # return
        move    $2,$5            # with result
```

Power / PowerPC

History
- **IBM develops Power architecture**
  - Basis of RS6000
  - Somewhere between RISC & CISC
- **IBM / Motorola / Apple combine to develop PowerPC architecture**
  - Derivative of Power
  - Used in Power Macintosh

CISC-like features
- **Registers with control information**
  - Set of condition registers (CR0–7) holding outcome of comparisons
  - link register (LR) to hold return PC
  - count register (CTR) to hold loop count
- **Updating load / stores**
  - Update base register with effective address

PPC rfact

Registers
- **r3**
  - argument n
  - return val
- **r31 saved arg**
- **CR0 comparison**

Instructions
`mfspr` move from special purp. reg.

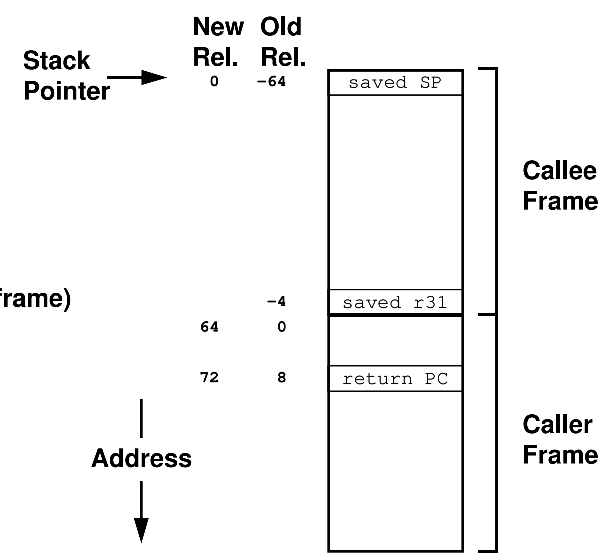
`cmpi` compare immediate

`cal` add immediate

`stu` store with update

```
.rfact:
        st      r31,-4(SP)      # save R31
        mfspr   r0,LR           # r0 <- return PC
        cmpi    0,r3,1          # compare n to 1
        st      r0,8(SP)        # save return PC
        cal     r31,0(r3)       # r31 <- n
        stu     SP,-64(SP)     # save & update SP
        ai      r3,r3,-1        # r3 <- n-1
        # if comparison result is >, goto __Le8
        bc      BO_IF,CR0_GT,__Le8
        l       r31,60(SP)      # restore r31
        cal     r3,1(r0)        # return val <- 1
        ai      SP,SP,64        # restore SP
        bcr     BO_ALWAYS,CR0_LT   # return
__Le8:
        bl      .rfact          # call rfact
        muls    r3,r3,r31       # compute result
        l       r12,72(SP)      # r12 <- return PC
        ai      SP,SP,64        # restore SP
        mtspr   LR,r12          # set return PC
        l       r31,-4(SP)      # restore R31
        bcr     BO_ALWAYS,CR0_LT   # return
```


PPC rfact Stack Frame



Frame
- **64 bytes**
- **old SP**
- **return PC (in caller's frame)**
- **R31**

PPC ifact

Registers
- **r3**
  - argument n
  - result
- **r4 i**
- **CR0 comparison**
- **CTR counter**

Stack Frame
- **none**

```
.ifact:
#       r3      argument n & result
#       r4      i
        cmpi    0,r3,2  # compare n to 2
        ai      r0,r3,-1        # r0 <- n-1
        cal     r3,1(r0)        # result <- 1
        cal     r4,2(r0)        # i <- 2
        bcr     BO_IF,CR0_LT    # return if n < 2
        mtspr   CTR,r0      # Count Reg <- n-1
__L8c:
        muls    r3,r4,r3        # result *= i
        ai      r4,r4,1         # i++
        # decrement & loop if nonzero
        bc      BO_dCTR_NZERO,CR0_LT,__L8c
        bcr     BO_ALWAYS,CR0_LT # return
```

VAX

Pinnacle of CISC
- **Maximize instruction density**
- **Provide instructions closely matched to typical program operations**

Instruction format
- **OP, arg1, arg2, …**
  - Each argument has arbitrary specifier
  - Accessing operands may have side effects

Condition Codes
- **Set by arithmetic and comparison instructions**
- **Basis for successive branches**

Procedure Linkage
- **Direct implementation of stack discipline**

VAX Operand Specifiers

Examples

Notation	Value	Side Eff	Use
Ri	ri		General purpose register
$v	v		Immediate data
(Ri)	M[ri]		Memory reference
v(Ri)	M[ri+v]		Mem. ref. with displacement
A[Ri]	M[a+ri*d]		Array indexing

- A is specifier denoting address a
- d is size of datum

Notation	Value	Side Eff	Use
(Ri)+	M[ri]	Ri += d	Stepping pointer forward
-(Ri)	M[ri-d]	Ri -= d	Stepping pointer back
@(Ri)+	M[M[ri]]	Ri += 4	Step through pointer table

VAX Registers

- **R0–R11 General purpose**
  - R2–R7 Callee save in example code
  - Use pair to hold double
- **R12 AP Argument pointer**
  - Stack region holding procedure arguments
- **R13 FP Frame pointer**
  - Base of current stack frame
- **R14 SP Stack pointer**
  - Top of stack
- **R15 PC Program counter**
  - Used to access data in code

VAX Procedure Linkage

Caller
- **Push Arguments**
  - Relative to SP
- **Execute CALLS n, proc**
  - n denotes number of arguments
  - proc starts with mask denoting registers to be saved on stack

CALLS Instruction
- **Creates stack frame**
  - Saved registers, PC, FP, AP, mask, PSW
- **Sets AP, FP, SP**

Callee
- **Compute return value in R0**
- **Execute ret**
  - Undoes effect of CALLS

VAX rfact

Registers
- r6 saved n
- r0 return value

Stack Frame
- save r6

Note
- Desination argument last

```
_rfact:
        .word 0x40       # Save register r6
        movl 4(ap),r6    # r6 <- n
        cmpl r6,$1       # if n > 1
        jgtr L1          # then goto L1
        movl $1,r0       # r0 <- 1
        ret              # return
L1:
        pushab -1(r6)    # push n-1
        calls $1,_rfact  # call recursively
        mull2 r6,r0      # return result * n
        ret
```


VAX ifact

Registers
- r0 result
- r1 i
- r2 n

Stack Frame
- No saved regs.

```
_ifact:
        .word 0x0        # no saved regs.
        movl 4(ap),r2    # r2 <- n
        movl $1,r0       # result <- 1
        movl $2,r1       # i <- 2
        cmpl r1,r2       # if i > n
        jgtr L3          # then goto L3
L5:
        mull2 r1,r0      # result *= i
        incl r1          # i++
        cmpl r1,r2       # if i > n
        jleq L5          # then goto L5
L3:
        ret              # return
```


Sparse Matrix Code

Task
- **Multiply sparse matrix times dense vector**
  - Matrix has many zero entries
  - Save space and time by keeping only nonzero entries
  - Common application

Compressed Sparse Row Representation

$$\begin{bmatrix} 3.5 & 0.9 & & 2.2 & & \\ & 4.1 & & 1.9 & & \\ 4.6 & & 0.7 & 2.7 & & 3.0 \\ & & & 2.9 & & \\ & & 1.2 & & 2.8 & \\ & & & & & 3.4 \end{bmatrix}$$

```
[(0,3.5) (1,0.9) (3,2.2)]
[(1,4.1) (3,1.9)]
[(0,4.6) (2,0.7) (3,2.7) (5,3.0)]
[(2,2.9)]
[(2,1.2) (4,2.8)]
[(5,3.4)]
```


CSR Encoding

Parameters
- **type** [Allows different formats]
- **nrow** Number of rows (and columns)
- **nentries** Number of nonzero matrix entries

Val
- **List of nonzero values (nentries)**

Cindex
- **List of column indices (nentries)**

Rstart
- **List of starting positions for each row (nrow+1)**

```
typedef struct {
  matrix_t type;
  int nrow;
  int nentries;
  double *val;
  int *cindex;
  int *rstart;
} csr_rec, *csr_ptr;
```

CSR Example

Parameters

- nrow = 6
- nentries = 13

Val

```
[3.5, 0.9, 2.2, 4.1, 1.9, 4.6, 0.7, 2.7, 3.0, 2.9, 1.2, 2.8, 3.4]
```

Cindex

```
[0,   1,   3,   1,   3,   0,   2,   3,   5,   2,   2,   4,   5]
```

Rstart

```
[0, 3, 5, 9, 10, 12, 13]
```

```
[(0,3.5)  (1,0.9)  (3,2.2)]
[(1,4.1)  (3,1.9)]
[(0,4.6)  (2,0.7)  (3,2.7)  (5,3.0)]
[(2,2.9)]
[(2,1.2)  (4,2.8)]
[(5,3.4)]
```


CSR Multiply: Clean Version

```
void csr_mult_smpl(csr_ptr M, double *x, double *z)
{
  int r, ci;
  for (r = 0; r < M->nrow; r++) {
    z[r] = 0.0;
    for (ci = M->rstart[r]; ci < M->rstart[r+1]; ci++)
      z[r] += M->val[ci] * x[M->cindex[ci]];
  }
}
```


CSR Multiply: Fast Version

```
void csr_mult_opt(csr_ptr M, ftype_t *x, ftype_t *z)
{
  ftype_t *val = M->val;
  int *cindex_start = M->cindex;
  int *cindex = M->cindex;
  int *rnstart = M->rstart+1;
  ftype_t *z_end = z+M->nrow;

  while (z < z_end) {
    ftype_t temp = 0.0;
    int *cindex_end = cindex_start + *(rnstart++);
    while (cindex < cindex_end)
      temp += *(val++) * x[*cindex++];
    *z++ = temp;
  }
}
```

Performance

- Approx 2X faster
- Avoids repeated memory references

PPC Inner Loop

Registers

- r3 *cip
- r4 x
- r10 valp-8
- r11 cip
- fp3 temp

```
while (...)
  temp += *(valp++) * x[*cip++];
```

```
__L208:
        rlinm   r3,r3,3,0,28     # *cip * 8
        lfdx    fp0,r4,r3       # fp0 <- x[*cip]
        lfdu    fp1,8(r10)       # fp1 <- *(++valp)
        lu      r3,4(r11)        # r3  <- *++cip
        fma     fp3,fp1,fp0,fp3 # fp3 += fp1*fp0
        # Decrement & loop
        bc      BO_dCTR_NZERO,CR0_LT,__L208
```

VAX Inner Loop

Registers

- r4 cip
- r2,r3 temp
- r5 valp
- r6 cip_end
- r10 x

```
    while (...)
      temp += *(valp++) * x[*cip++];
```

```
L36:
        movl  (r4)+,r0    # r0 <- *cip++
        muld3  (r5)+,(r10)[r0],r0 # r0,r1 <- *valp++ * x[r0]
        addd2  r0,r2      # temp += r0,r1
        cmpl  r4,r6       # if cip < cip_end
        jlssu L36         # then goto L36
```


Performance Comparison

Experiment

- 10 X 10 matrices
- 100% density
- 100 multiply accumulates

Machine	MHz	µsecs	Cycles/Ele
MIPS	25	365	18
PPC 601	62	63	8
PPC 603	75	125	19
VAX	25?	2448	122?

Summary

MIPS

- Simple register state
- Every operation has single effect
  - Load, store, operate, branch

VAX

- Hidden control state
- Operations vary from simple to complex
- Side effects possible

Power PC

- Complex control state
- Operations simple to medium
- Side effects possible
- Hard target for code generator