Honors Compilers

An Introduction to MMIX

Jan 31st 2002
MMIX – The General Idea

- MMIX is intended as a realistic RISC machine that can be used for analyzing efficiency of code at a low level
- Very uniform and clean structure
- Ideal for compilers
Basic architecture

- 256 registers, each 64-bits
  - Completely uniform register usage
  - Not even $R0 is special
- Big-endian addressing
- 64-bit addresses, byte addressable
### Instruction Formats

- **All instructions are four bytes (32 bits)**
  - OP  X  Y  Z
    - 8-bits  8-bits 8-bits 8-bits
- **Some instructions combine YZ**
  - OP  X  YZ
    - 8-bits 8-bits 16-bits
- **Some instructions combine XYZ**
  - OP  XYZ
    - 8-bits 24-bits
Instruction Formats

- Most often, X,Y,Z are register numbers
  - ADD $2,$5,$6 means $2 := $5 + $6
- Z may be an immediate value
  - ADD $2,$5,6 means $2 := $5 + 6
- Assembler selects opcode based on type of Z, e.g. ADD reg is 32, ADD immed is 33
- The notation $Z/Z means that an operand can be either a register or immediate
Data Types

- One nibble = 4 bits (hex or decimal digit)
- One byte = 2 nibbles = 8 bits
- One wyde = 2 bytes = 16 bits
- One tetra = 2 wydes = 32 bits
- One octa = 2 tetras = 64 bits
- Integer values are unsigned, or 2’s complement signed values. Make sure you understand 2’s complement!
Load instructions

- **LDB** $X,$Y,$Z/Z  (Load Byte)
  - $X$ loaded from memory address $Y+$Z/Z
  - This is a signed load, with sign extension
- **LDBU** $X,$Y,$Z/Z  (Load Byte Unsigned)
  - Same but unsigned with zero extension
- Similarly for LDW[U] LDT[U] LDO[U]
  - To load wyde/tetra/octa signed/unsigned
- **LDHT** loads high tetra, low tetra zeroed
Store instructions

- **STB $X,$Y,$Z/Z** (Store Byte)
  - $X$ stored to memory address $Y+$Z/Z
  - This is a signed load, with overflow check
- **STBU $X,$Y,$Z/Z** (Store Byte Unsigned)
  - Same but unsigned with no overflow check
- Similarly for **STW[U] STT[U] STO[U]**
  - To store wyde/tetra/octa signed/unsigned
- **STCO X,$Y,$Z/Z**
  - Stores octabyte with unsigned value $X$
- **STHT** (Store high tetra)
Addition and Subtraction

- **ADD $X,$Y,$Z/Z (Add)**
  - Sets $X$ to $Y+$Z/Z, with overflow check

- **ADDU $X,$Y,$Z/Z (Add unsigned)**
  - Same but with no overflow check

- **2ADDU,4ADDU,8ADDU,16ADDU**
  - Same as ADDU, but $Y * 2/4/8/16$

- **SUB/SUBU (Subtract, subtract unsigned)**

- **NEG/NEGU $X,Y,$Z/Z**
  - Like SUB/SUBU but Y is immediate
Bit Fiddling

- AND/OR/XOR $X,$Y,$Z/Z
  - ANDN (And not)
  - ORN (Or not)
  - NAND (not and)
  - NOR (not or)
  - NXOR (not xor)
- Plus some exotic instructions for graphics
Immediate Wyde Instructions

- **SETH $X, YZ**  
  sets left 16 bits of $X
  - SETMH  sets next 16 bits of $X
  - SETML  sets next 16 bits of $X
  - SETL   sets last 16 bits of $X
  - (remaining bits set to zero)

- **INCH/MH/ML/L** – add wyde

- **ORH/MH/ML/L** – or wyde

- **ANDNM/MH/ML/L** – and not wyde
Shift Instructions

- **SL $X,$Y,$Z/Z (shift left)**
  - Shift $Y$ left by $Z/Z$ bits, with overflow check
- **SLU (Shift left unsigned)**
  - Same but not overflow check
- **SR (Shift right)**
  - Shifts right with sign extension
- **SRU (Shift right unsigned)**
  - Shifts right with zero fill
Comparison Instructions

- **CMP $X,$Y,$Z/Z (Compare)**
  - Signed comparison of $Y with $Z/Z
  - $X set to -1,0,+1 for less,equal,greater

- **CMPU $X,$Y,$Z/Z (Compare unsigned)**
  - Same but comparison is unsigned
Conditional Set Instructions

- CSN $X,Y,Z/Z$ (conditional set if neg)
  - If $Y$ is negative, $X$ set to $Z/Z$, else nop

- Similarly for:
  - zero (Z) positive (P) odd (OD)
  - nonnegative (NN) nonzero (NZ)
  - Z/Z (zero or set if negative)
Conditional Branches

- **BN $X,YZ** (branch if negative)
  - YZ interpreted as count of instructions
  - Each instruction is four bytes (one tetra)
  - Two forms of each branch, forward/backward
  - 8 opcodes for 8 standard conditions

- **PBN $X,YZ** (probable branch if negative)
  - Same, but predicted to be more likely to branch, which affects efficiency only.

- **GETA $X,YZ** (sets $X to jump address)
Jump Instructions

- **JMP XYZ** (unconditional jump)
  - Two forms forward/backward

- **GO $X,$Y,$Z/Z**
  - Jumps to instruction at $Y+$Z/Z
  - $X$ is set to address of GO + 4
Multiply and Divide

- **MUL $X,$Y,$Z/Z (multiply)**
  - Signed multiply with overflow check
  - MULU, same but no overflow check
    - High order 64 bits goes into special register rH

- **DIV $X,$Y,$Z/Z**
  - Signed divide, remainder to rR

- **DIVU $X,$Y,$Z/Z**
  - rD & $Y$ divided by $Z/Z$
  - Quotient to $X$, remainder to rR
Subroutine Linkage

- Fancy instructions with register windows
  - PUSHJ/PUSHGO
- But let’s keep things more typical by using
  - GO $X,$Y,0
    - $X$ is linkage register, $Y$ contains address
  - GETA $X$, YZ
    - Here address of routine is YZ from GETA
  - GO $Y,$Y,0
    - Is the return instruction
Caching Instructions

- **LDUNC** (load octa uncached)
- **STUNC** (store octa uncached)
- **PRELD X,$Y,$Z/Z** (preload data in cache)
  - Data from $M[Y+Z/Z] .. M[Y+Z/Z+X]$ will be used in near future
- **PREST**
  - Data will be stored before it is loaded
End of Lecture

- Assignment
- Get access to simulator
- Write a trivial program
- Run under simulator
- This is simply a test of access
- See you on Tuesday