Processor Design
–
Instruction Coding

Professor Jari Nurmi
Institute of Digital and Computer Systems
Tampere University of Technology, Finland
email jari.nurmi@tut.fi

Instruction Coding for Dummies

- Part of the instruction set design done in requirement capture
- Here we concentrate on coding the operations, operands and addressing modes in the instruction words
- The targets of the instruction coding include
  - correct and distinguishable codes for all instructions (!)
  - low cost = small memory footprint of the programs
  - low power consumption
  - low complexity of the decoder
  - straightforward memory interface
  - ease of assembling programs
  - create good target for compilers
  - good performance of instructions
  - good command over the processor resources
- The targets are in most cases contradictory to each other

Alternatives to Achieve the Targets

- Horizontal coding
  - low complexity of decoder
  - good performance
  - good command over processor resources
- Vertical coding
  - small memory footprint
  - low power
  - relatively low complexity of decoder, if FIELD-BASED
- Variable instruction size
  - small memory footprint
  - good command over processor resources
- Fixed instruction size (or a couple of fixed alternatives)
  - straightforward memory interface
  - low complexity of the decoder (and program control)

Typical Choices

- Division of instruction word to fields
  - Simple decoding
  - Easy assembling
- Short fixed-length instructions
  - Low-cost, low-power
  - Simple decoding
  - Straightforward memory interface
- Compromises to accommodate more instructions
  - Deviation from the fixed fields
  - Special bit combinations to encode some special cases
  - Use of two lengths of instructions (like ARM Thumb, MIPS 16, ...)
  - Deviation from general purpose register usage

Coding Example

- 16-bit processor architecture
- 16 general-purpose registers
- 2's complement data
- Code the following 21 operations
  - single-cycle operations
  - single 16-bit instruction for each operation

INSTRUCTION	ADDRESSING MODES, LIMITATIONS, ETC.
Add	2 source and 1 destination registers
Add with carry	2 source and 1 destination registers
Subtract	2 source and 1 destination registers
Multiply	Mixture of signed and unsigned operands. 2 source and 1 **double** destination regs.
Logical shift	1 src, 1 dest reg, ±15 bit shifts from instruction
Logical shift on register	1 src, 1 dest reg, 1 shift-amount register
AND	2 source and 1 destination registers
OR	2 source and 1 destination registers
XOR	2 source and 1 destination registers
NOT	1 source and 1 destination registers
Conditional branch	Up to 7 conditions, PC-relative branch in ±255 range
Unconditional branch	PC-relative branch in ±255 range
Conditional branch on register	Up to 7 conditions, 1 reg. for branch address
Unconditional branch on register	1 reg. for branch address
Conditional branch and link on register	Up to 7 conditions, 1 reg. for branch address, 1 link register
Unconditional branch and link on register	1 reg. for branch address, 1 link register
Load	Register indirect with possible post-inc/dec, 1 dest reg.
Store	Register indirect with possible post-inc/dec, 1 src reg.
Load immediate	Signed ±127 immediate, 1 dest reg.
Register – register move	1 src and 1 dest reg.
No operation	

- Start from the regular operations
  - 3-register operations ADD, ADDC, SUB, LSHR, AND, OR, XOR
    - 7 operations (need < 3 bits of the opcode field)
    - 3 register fields for each (4 bits)

0xxx	REG	REG	REG

  - Then 2-register operations NOT, LD, ST, MOVE
    - 2 register fields
    - space for additional info (LD/ST direction, post modif., Reg. move, not)
    - use the 8th 3-bit operation code for these

0111	4 bit specifier	REG	REG

    - the specifier first 2 bits: LD, ST, MOVE, NOT (01, 10, 11, 00)
    - the specifier last 2 bits: increment, decrement, no change (10, 01, 00)

- The remaining ones then
  - Multiplication (signed/unsigned information + double registers)
    - two source registers = two fields
    - signed/unsigned indication of both operands = 2 bits
    - double register – if we restrict to pairs 1 & 0, 3 & 2, ... we need 3 bits

100	s/u	s/u	REG PAIR	REG	REG

    - we had to ”borrow” from the opcode field
    - combinations US and SU are redundant
  - Logical shift with immediate
    - 5-bit signed immediate for shift value
    - 2 fields for source and destination registers
    - again, borrowing from the opcode

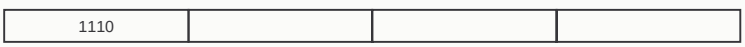
101		SHIFT	REG	REG

- And still...
  - Branches
    - with 9-bit immediate offset (and conditions)
    - 7 conditions + unconditional branch ❍ 3 bits needed

    - on register (and conditions)
    - branch and link on register (and conditions)

  - Load immediate
    - 8-bit immediate value

- Finally,
  - NOP
    - several options as a macro instruction
      - MOVE register to itself
      - OR R0, R0, R0 (could be coded 0x0000)
      - AND R0, R0, R0 (also possible to set 0x0000)
      - JUMP to the next instruction
      - the drawback is the power consumed in executing NOP
    - or actual binary code distinguished by the instruction decoder
      - LSH with shift –16 (0b10000) decoded from the 8 MSB bits
      - as a two-register instruction with both inc and dec bit set (decoded from 6 bits)
      - the remaining opcode 0b1111 (decoded from the 4 MSB bits)



Overview of the Instruction Coding

[15:12]	[11:8]	[7:4]	[3:0]	interpretation		[15:12]	[11:8]	[7:4]	[3:0]	interpretation	
0000	*reg*	*reg*	*reg*	ADD		**1000**	**0**RRR	*reg*	*reg*	MUL_UU	
0001	*reg*	*reg*	*reg*	ADDC			**1**RRR	*reg*	*reg*	MUL_US	
0010	*reg*	*reg*	*reg*	LSHR		**1001**	**0**RRR	*reg*	*reg*	MUL_SU	
0011	*reg*	*reg*	*reg*	SUB			**1**RRR	*reg*	*reg*	MUL_SS	
0100	*reg*	*reg*	*reg*	AND		**101**s	ssss	*reg*	*reg*	LSH	
0101	*reg*	*reg*	*reg*	OR		**1100**	ccci	iiii	iiii	Bccc	
0110	*reg*	*reg*	*reg*	XOR			**111**i	iiii	iiii	B	
0111	**00**xx	*reg*	*reg*	MOVE		**1101**	ccc**0**	xxxx	*reg*	BRccc	
	01+-	*reg*	*reg*	ST			**1110**	xxxx	*reg*	BR	
	10+-	*reg*	*reg*	LD			ccc**1**	*reg*	*reg*	BRALccc	
	11xx	*reg*	*reg*	NOT			**1111**	*reg*	*reg*	BRAL	
						1110	iiii	iiii	*reg*	LDC	
						1111	xxxx	xxxx	xxxx	NOP	

End of instruction coding

next we will look at operating system
(and software in general)
impacts on processor design