ECE 571 – Advanced Microprocessor-Based Design Lecture 3

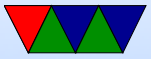
Vince Weaver

http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

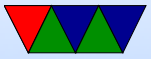
22 January 2013

The ARM Architecture



Brief ARM History

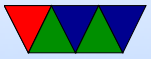
- ACORN
- Wanted a chip to succeed 6502. Decided to make one themselves. (Good idea, 65816 a pain to program and only 16-bit)



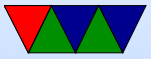
Models

Architecture vs Family

- ARMv1 : ARM1
- ARMv2 : ARM2, ARM3 (26-bit, status in PC register)
- ARMv3 : ARM6, ARM7
- ARMv4 : StrongARM, ARM7TDMI, ARM9TDMI
- ARMv5 : ARM7EJ, ARM9E, ARM10E, XScale



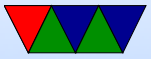
- ARMv6 : ARM11, ARM Cortex-M0 (Raspberry Pi)
- ARMv7 : Cortex A8,A9,Cortex-M3 (iPad, iPhone, Pandaboard)
- ARMv8 : Cortex A-50 (64-bit)



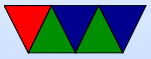
Various abbreviations in Model Names

For example, ARM7DTMI

- "Application" ARM Cortex-A
- "Real-time" ARM Cortex-R
- "Micro-controller" ARM Cortex-M
- "E" means DSP instructions
- "M" improved multiplier

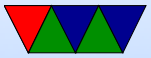


- “T” THUMB
- “J” Jazelle (java bytecodes)
- D/I Debug/ICE
- EE – ThumbExecutionEnvironment, Just-in-time
- NEON – SIMD
- VFP – Floating point

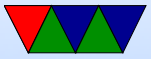


ARM Architecture

- 32-bit
- Load/Store
- Can be Big-Endian or Little-Endian (usually little)
- Fixed instruction width (32-bit, 16-bit THUMB) (Thumb2 is variable)
- Opcodes take three arguments (Destination, Source, Source)

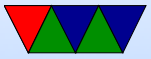


- Cannot access unaligned memory (optional newer chips)
- Conditional execution
- Status flag (many instructions can optionally set)
- Complicated addressing mode
- Most features optional (FPU [except in newer], PMU, Vector instructions, Java instructions, etc.)

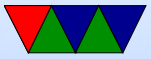


Registers

- Has 16 GP registers (more available in supervisor mode)
- r0 - r12 are general purpose
- r13 is stack pointer (sp)
- r14 is link register (lr)
- r15 is program counter (pc) (reading r15 usually gives PC+8)

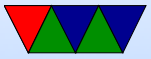


- 1 status register (more in system mode).
  NZCVQ (Negative, Zero, Carry, oVerflow, Saturate)

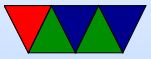


Arithmetic Instructions

- adc, adcs – add with carry
- add, adds – add
- adr – add immediate to PC, store address in reg
- cmp, cmn – compare, compare negative
- mul – multiply two registers, only least sig 32bit saved
- rsb, rsbs – reverse subtract (immediate - rX)

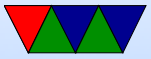


- rsc, rscs – reverse subtract with carry
- sbc, sbcs – subtract with carry
- sub, subs – subtract



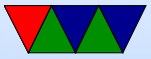
Shift Instructions

- asr, asrs – arith shift right
- lsl – logical shift left
- lsr – logical shift right
- ror, rors – rotate right
- rorx – rotate right extend, carry flag shift into bit 31

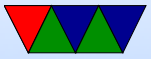


Logic Instructions

- and, ands – bitwise and
- bfc – bitfield clear, clear bits in reg
- bfi – bitfield insert
- bic, bics – bitfield clear: and with negated value
- clz – count leading zeros (armv7)
- eor, eors – exclusive or (name shows 6502 heritage)

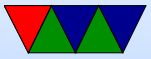


- orn, orns – or not (armv6)
- orr, orrs – bitwise or
- tst – test (and with value, don't save)



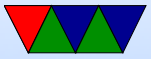
Control-Flow Instructions

- b – branch (can use all the condition code prefixes)
- bl, blx – branch and link (X means change to thumb)
- bx – branch and exchange to thumb
- mov pc,lr

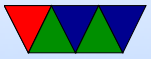


Load/Store Instructions

- ldr – load register
- ldrb – load register byte
- ldrd – load double, into consecutive registers
- ldrh – load register halfword, zero extends
- ldrsb – load register signed byte, sign-extends
- ldrsh – load register halfword, sign-extends

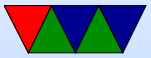


- str – store register
- strb – store byte
- strd – store double
- strh – store halfword



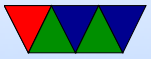
Register Manipulation

- mov, movs – move register
- movt – write value to top 16 bits of reg (armv6)
- mvn, mvns – move inverted
- nop – no-operation
- pli etc – preload instructions



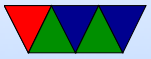
Stack Manipulation

- ldm, ldmia – load multiple memory locations into consecutive registers
- stm – store multiple, can be used like a PUSH instruction
- pop – pop from stack. specify a list of registers
- push – push multiple registers



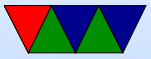
System Instructions

- svc, swi – software interrupt
- swap – swap value between register and memory (deprecated armv7)

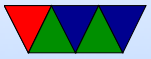


Fancy ARMv6

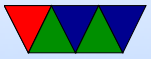
- mla – multiply/accumulate (armv6). multiply two add in one
- mls – multiply and subtract
- pkh – pack halfword (armv6)
- qadd, qsub, etc. – saturating add/sub (armv6)
- rbit – reverse bit order (armv6)



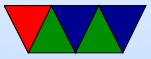
- rbyte – reverse byte order (armv6)
- rev16, revsh – reverse halfwords (armv6)
- sadd16 – do two 16-bit signed adds (armv6)
- sadd8 – do 4 8-bit signed adds (armv6)
- sasx – (armv6)
- sbfx – signed bit field extract (armv6)
- sdiv – signed divide (only armv7-R)



- udiv – unsigned divide (armv7-R only)
- sel – select bytes based on flag (armv6)
- sm* – signed multiply/accumulate
- setend – set endianess (armv6)
- sxtb – sign extend byte (armv6)
- tbb – table branch byte, jump table (armv6)
- teq – test equivalence (armv6)

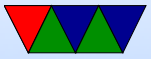


- u* – unsigned partial word instructions



Orthogonality of PC

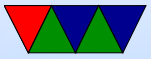
- Can branch with a “move” instruction
- Can do PC relative addressing easily
- Neat feature but complicates modern chip design



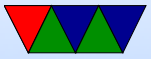
Prefixed instructions

Most instructions can be prefixed with condition codes:

- EQ, NE (equal)
- MI, PL (minus/plus)
- HI, LS (unsigned higher/lower)
- GE, LT (greaterequal/lessthan)
- GT, LE (graeterthan, lessthan)

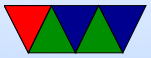


- CS, CC (carry set/clear)
- VS, VC (overflow set / clear)
- AL (always)



Setting Flags

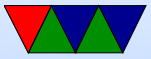
- `add r1,r2,r3`
- `adds r1,r2,r3` – set condition flag
- `addeqs r1,r2,r3` – set condition flag and prefix compiler and disassembler like `addseq`, GNU as doesn't?



Conditional Execution

```
if (x == 1 )
    a+=2;
else
    b-=2;
```

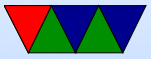
```
cmp     r1, #5
addeq   r2,r2,#2
subne   r3,r3,#2
```



Extra Shift in ALU instructions

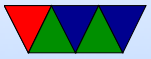
If second source is a register, can optionally shift:

- LSL – Logical shift left
- LSR – Logical shift right
- ASR – Arithmetic shift right
- ROR – Rotate Right
- RRX – Rotate Right with Extend



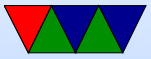
- For example:

```
add r1, r2, r3, lsl #4
r1 = r2 + (r3<<4)
```

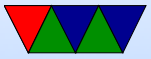


Addressing Modes

- `ldrb r1, [r2]` @ register
- `ldrb r1, [r2,#20]` @ register/offset
- `ldrb r1, [r2,+r3]` @ register + register
- `ldrb r1, [r2,-r3]` @ register - register
- `ldrb r1, [r2,r3, LSL #2]` @ register +/- register, shift



- `ldrb r1, [r2, #20]!` @ pre-index. Load from r2+20 then write back
- `ldrb r1, [r2, r3]!` @ pre-index. register
- `ldrb r1, [r2, r3, LSL #4]!` @ pre-index. shift
- `ldrb r1, [r2],#+1` @ post-index. load, then add value to r2
- `ldrb r1, [r2],r3` @ post-index register
- `ldrb r1, [r2],r3, LSL #4` @ post-index shift



Kernel Programming ABIs

- OABI – "old" original ABI (arm). Being phased out. slightly different syscall mechanism, different alignment restrictions
- EABI – new "embedded" ABI (armel)
- hard float – EABI compiled with VFP (vector floating point) support (armhf)