MANCHEstER
1824

The University
of Manchester

Architectural extensions

- Outline:
  - instruction set extensions
  - digital signal processing instructions
  - security extensions
  - Java support
  - future instruction set developments

☞ hands-on: Thumb C and cycle counts

Architectural extensions

- Outline:
  - ➜ **instruction set extensions**
  - ❍ digital signal processing instructions
  - ❍ security extensions
  - ❍ Java support
  - ❍ future instruction set developments

☞ hands-on: Thumb C and cycle counts

Instruction set extensions

- Since its introduction the ARM instruction set has been extended several times
  - extensions to v4 have been included already
    - e.g. halfword support, Thumb, …
  - v5, v5TE and v6 extensions are described here
    - better ARM/Thumb interworking
    - more ‘endian’ support
    - variety of minor enhancements
    - DSP support – in following subsection

MANCHEStER 1824

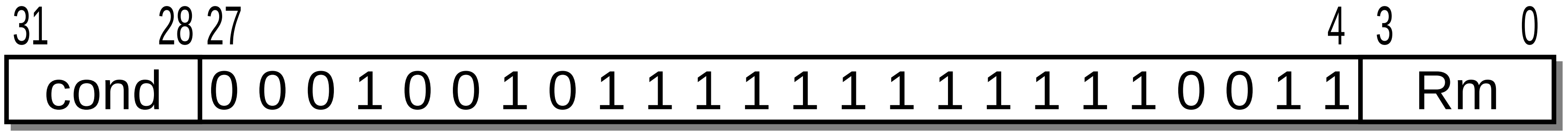
The University of Manchester

Instruction set extensions – v5

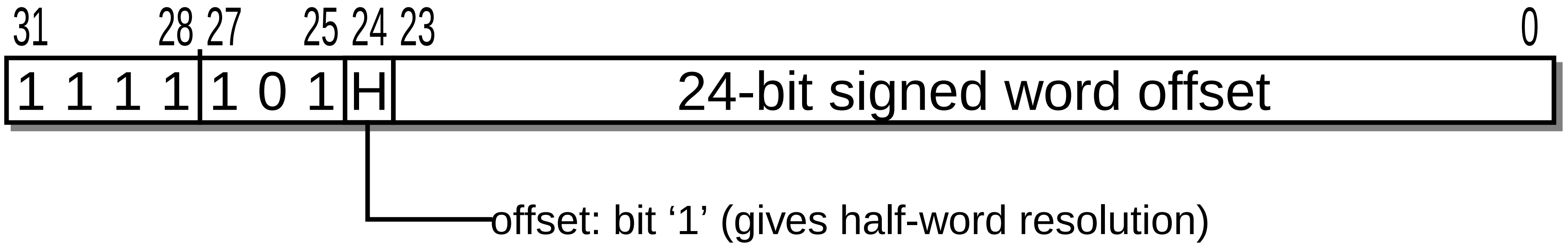
- BLX
  - Branch with Link and eXchange
- CLZ
  - Count Leading Zeros
- BKPT
  - software breakpoint
- PLD
  - Cache PreLoaD
- Extra coprocessor op-codes
  - CDP2, MRC2, etc.

Instruction set extensions – v5

- BLX - two forms
  - BLX Rm



  - BLX label


31
28 27
25 24 23
0
1 1 1 1
1 0 1
H
24-bit signed word offset
offset: bit ‘1’ (gives half-word resolution)

    – Note: no condition code always executes

MANCHEStER 1824

The University of Manchester

Instruction set extensions – v5

- CLZ Rd, Rm

31 28	27 16	15 12	11 4	3 0
cond	0 0 0 1 0 1 1 0 1 1 1 1	Rd	1 1 1 1 0 0 0 1	Rm

  - Returns number of 0s from MSB (0-32)

- BKPT

31 28	27 20	19 8	7 4	3 0
1 1 1 0	0 0 0 1 0 0 1 0	12-bit immed	0 1 1 1	Rm

  - Allows user to force ‘prefetch abort’

MANCHEstER 1824

The University of Manchester

Instruction set extensions – v5

- PLD <addressing mode>

31 28	27 26	25	24	23	22 20	19 16	15 12	11 0
1 1 1 1	0 1	1	1	U	1 0 1	Rd	1 1 1 1	addressing mode

- PreLoaD
  - a hint to the memory that this address may be wanted, soon
- has no effect on the programmer-visible state
- may cause a cache line fetch
  - memory can choose to ignore this operation
- cannot generate aborts

MANCHEStER
1824

The University
of Manchester

Instruction set extensions – v5/v6

- Several instructions to support DSP
  - mostly multiply and multiply-accumulate
  - most dealt with shortly

- UMAAL (v6) is a long multiply with two accumulates

```
UMAAL  R2, R3, R1, R0
R3:R2 := (R1 X R0) + R2 + R3
```

  – encoded in the ‘normal’ multiply set

MANCHEstER
1824

The University
of Manchester

Instruction set extensions – v6

- ARM v6 has extra control operations:
  - CPS
    - Change Processor State (switch mode)
  - SETEND
    - Endian control bit appears in PSR
  - SRS/RFE
    - Save Return State (Push LR and SPSR)
    - Return From Exception (Pop PC and CPSR)

Semaphore operations

- All ARMs support "swap"

```
SWP    R1, R2, [R3]
```

- New operations from v6

```
LDREX  R0, [R1]
```

  – Load exclusive … TLB notes processor ID

```
STREX  R2, R0, [R1]
```

  – Store exclusive … fails if 'wrong' processor
  – R2 holds failure flag

MANCHEstER 1824

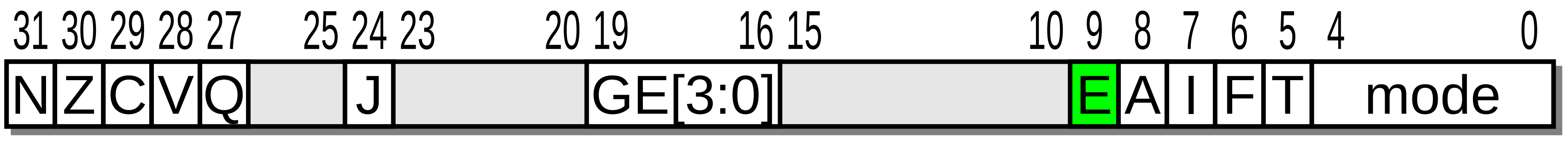
The University of Manchester

Instruction set extensions – v6

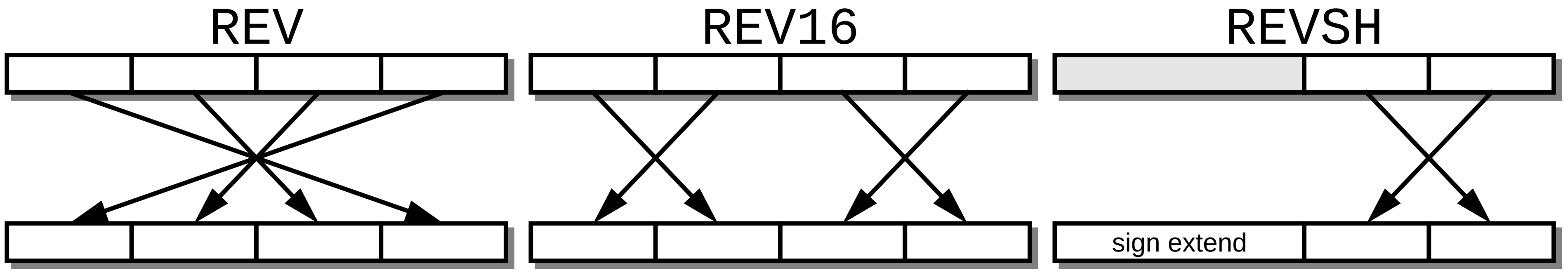
- Unaligned memory accesses
  - in earlier ARMs
    - word and halfword accesses *should* be appropriately aligned
      - unaligned accesses are ‘interesting’
    - misalignment *may* cause a trap (via MMU – see later)
  - on ARM v6
    - unaligned accesses are supported in hardware
    - still not a good idea!
      - may reduce performance

Instruction set extensions – v6

- Endian control
  - on ARM v6 **data ‘endianess’** is explicit in the CPSR



    - can be changed by `SETEND BE|LE` instructions
    - instructions are still ‘little endian’
    - can be modified by the MMU, if present
  - software also assisted by new instructions



MANCHEStER 1824

The University of Manchester

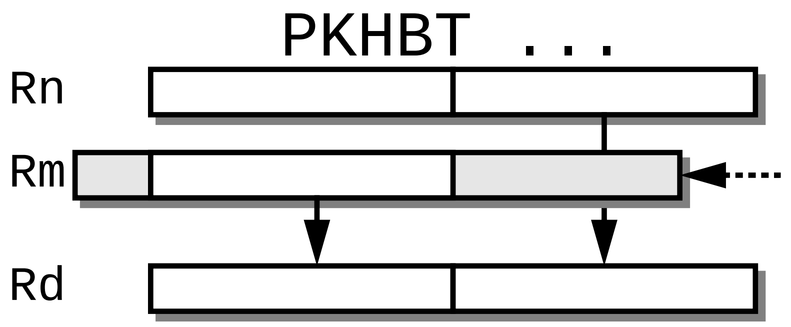
Instruction set extensions – v6

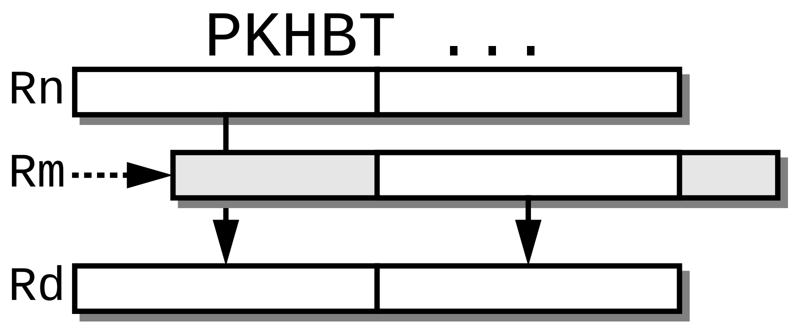
- Data packing
  - one 32-bit register may be used for two 16-bit variables
  - PKHBT Rd, Rn, Rm {, LSL #<0-31>}
  - PKHTB Rd, Rn, Rm {, ASR #<1-32>}

31–28	27–20	19–16	15–12	11–7	6	5–4	3–0
cond	0 1 1 1 1 0 0 0	Rn	Rd	shift	?	0 1	Rm

0 PKHBT
1 PKHTB

  - two 16-bit quantities are packed together (with optional shift)





MANCHEStER 1824

The University of Manchester

Architectural extensions

- ❑ Outline:
  - ❍ instruction set extensions
  - ➜ **digital signal processing instructions**
  - ❍ security extensions
  - ❍ Java support
  - ❍ future instruction set developments

☞ hands-on: Thumb C and cycle counts

MANCHEstER
1824

The University
of Manchester

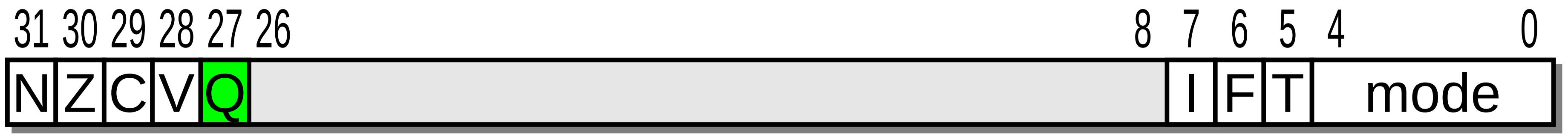
Digital signal processing

- Many ARM applications require good 16-bit signal processing performance
  - e.g. GSM mobile phone handset
- One solution is ARM plus separate DSP core
  - two software development toolkits
  - difficulty producing integrated solution
- ARM has offered two solutions:
  - Piccolo DSP coprocessor
    - little commercial take-up
  - instruction set extensions
    - began with v5TE; extended in v6

MANCHEStER 1824

The University of Manchester

v5TE signal processing extensions



- Q bit added to the CPSR (and SPSRs)
  - detects saturating arithmetic overflow
  - sticky:
    - set by overflow
    - reset only by an MSR instruction

v5TE signal processing extensions

- Multiply instructions:

```
SMLAWy{cond}  Rd,Rm,Rs,Rn
SMULWy{cond}  Rd,Rm,Rs
SMLALxy{cond} RdLo,RdHi,Rm,Rs
SMULxy{cond}  Rd,Rm,Rs
```

  - provide various 16x16 and 16x32 multiply and multiply-accumulate operations
    - 16-bit operand can be selected from low or high half of register
    - ‘x’ and ‘y’ (above) are ‘B’ or ‘T’ for Bottom or Top 16 bits

v5TE signal processing extensions

- Saturating arithmetic instructions:
  - 32-bit saturating add/subtract:

```
QADD{cond} Rd,Rm,Rn
QSUB{cond} Rd,Rm,Rn
```

  - 32-bit saturating double then add/subtract

```
QDADD{cond} Rd,Rm,Rn
QDSUB{cond} Rd,Rm,Rn
```

    – allows for coefficients > 1
    – as required by some common algorithms

v5TE signal processing extensions

- Example inner product:

```
loop   LDR    r1,[r6],#4     ; get next two multipliers
       LDR    r2,[r7],#4     ; get next 2 multiplicands
       SMULBB r3,r1,r2       ; 16x16 multiply
       QDADD  r5,r5,r3       ; saturating x2 accumulate
       SMULTT r3,r1,r2       ; 16x16 multiply
       QDADD  r5,r5,r3       ; saturating x2 accumulate
       SUBS   r4,r4,#2       ; decrement loop counter
       BNE    loop           ;
```

  - 32-bit loads use memory efficiently

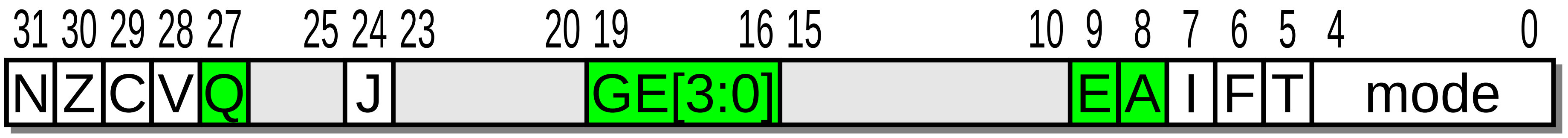
v5TE signal processing extensions

- Inner product - reordered:

```
        LDR     r1,[r6],#4      ; get first two multipliers
        LDR     r2,[r7],#4      ; get first 2 multiplicands
loop    SMULBB  r3,r1,r2        ; 16x16 multiply
        SUBS    r4,r4,#2        ; decrement loop counter
        QDADD   r5,r5,r3        ; saturating x2 accumulate
        SMULTT  r3,r1,r2        ; 16x16 multiply
        LDR     r1,[r6],#4      ; get next two multipliers
        QDADD   r5,r5,r3        ; saturating x2 accumulate
        LDR     r2,[r7],#4      ; get next 2 multiplicands
        BNE     loop            ;
```

  - instruction scheduling avoids pipeline stalls

Complete v6 PSR



- Q flag – saturating operation has saturated
- J flag – Java support (see later)
- GE flags (individual byte Greater than or Equal)
  - affected by SIMD arithmetic
  - used by SEL to *select* bytes/halfwords
- E flag – endianness of loads and stores (1 = big)
- A flag – disable imprecise aborts
  - precise aborts allow code to recover (e.g. from page fault)
  - … but keeping state may impair performance

MANCHEStER
1824

The University
of Manchester

v6 signal processing extensions

- The majority of the v6 DSP extensions are 'SIMD' operations
  - Single Instruction Multiple Data
    - Similar to Intel MMX
- SIMD add & subtract
    - two independent 16-bit operations, or
    - four independent -bit operations
  - operands may be signed or unsigned
  - in case of overflow
    - operations may set GE flags
    - results may saturate (and set Q flag)

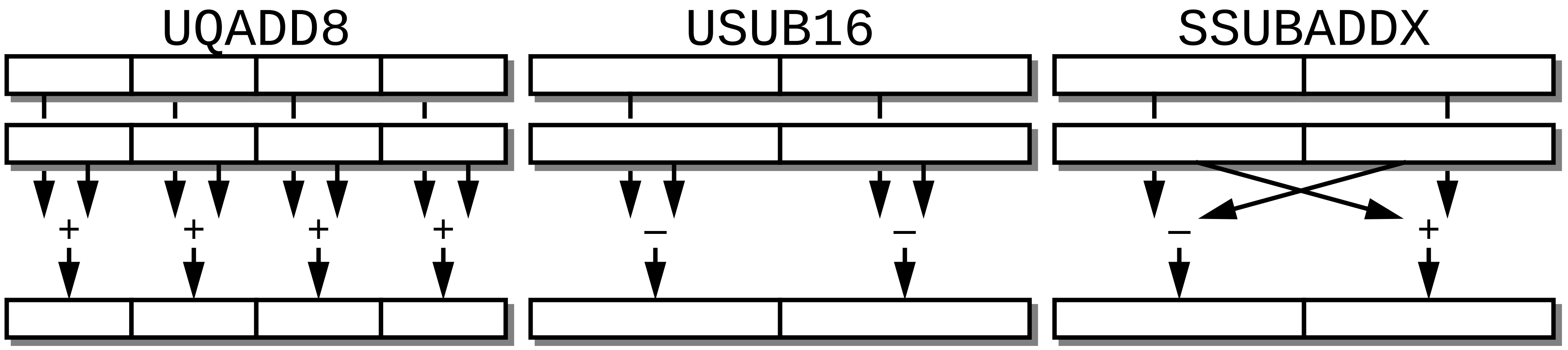
MANCHEstER 1824

The University of Manchester

v6 signal processing extensions

Saturating		Non-saturating		Data size	Operation
unsigned	signed	unsigned	signed		
UQADD8	QADD8	UADD8	SADD8	4 x 8-bit	add corresponding bytes in Rn and Rm
UQSUB8	QSUB8	USUB8	SSUB8	4 x 8-bit	subtract corresponding bytes in Rn and Rm
UQADD16	QADD16	UADD16	SADD16	2 x 16-bit	add corresponding halfwords in Rn and Rm
UQSUB16	QSUB16	USUB16	SSUB16	2 x 16-bit	subtract corresponding halfwords in Rn and Rm
UQADDSUBX	QADDSUBX	UADDSUBX	SADDSUBX	2 x 16-bit	halfword op. with Rm halves swapped then high halves added, low subtracted
UQSUBADDX	QSUBADDX	USUBADDX	SSUBADDX	2 x 16-bit	halfword op. with Rm halves swapped then high halves subtracted, low added

- examples:



  – also options to halve the result before writeback (e.g. UHSUB8)

MANCHEstER 1824

The University of Manchester

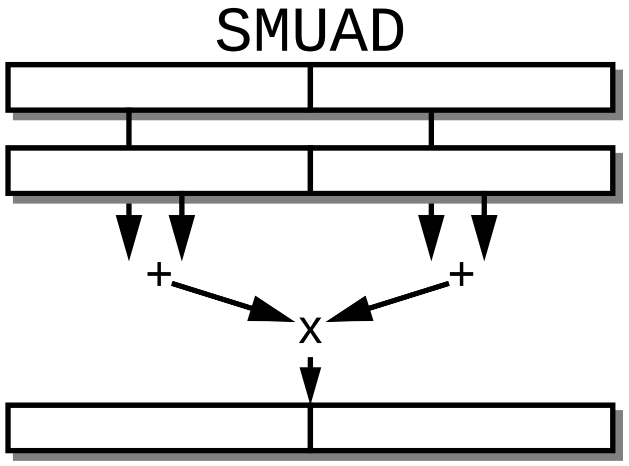
v6 signal processing extensions

- More 16 x 16 multiplies:

Instruction	Effect
`SMUAD Rd, Rm, Rs`	Rd :=$Rm_B \times Rs_B + Rm_T \times Rs_T$
`SMUSD Rd, Rm, Rs`	Rd :=$Rm_B \times Rs_B + Rm_T \times Rs_T$
`SMLAD Rd, Rm, Rs, Rn`	Rd := Rn + $Rm_B \times Rs_B + Rm_T \times Rs_T$
`SMLSD Rd, Rm, Rs, Rn`	Rd := Rn + $Rm_B \times Rs_B - Rm_T \times Rs_T$
`SMLALD RdLo, RdHi, Rm, Rs`	RdHi:RdLo := RdHi:RdLo + $Rm_B \times Rs_B + Rm_T \times Rs_T$
`SMLSLD RdLo, RdHi, Rm, Rs`	RdHi:RdLo := RdHi:RdLo + $Rm_B \times Rs_B - Rm_T \times Rs_T$

– ‘T’ and ‘B’ indicate the Top and Bottom halves of the register

– an ‘X’ can be added which swaps the halfwords of Rs first



MANCHEstER 1824

The University of Manchester

v6 signal processing extensions

- Support instructions
  - sign extend/zero extend

```
e.g.    UXTB    Rd, Rm  ; zero extend byte
        SXTB16  Rd, Rm  ; sign ext. 2 bytes ⇒ 2 halfwords
```

- on 8- or 16-bit quantities
- with optional rotation (8, 16, 24 places) first
- with optional subsequent accumulate

  - saturate

```
e.g.    SSAT    Rd, #n, Rm  ; signed saturation to n bits
```

- saturate (if necessary) to specified size (in bits)
- also allows preceding shift

MANCHEStER 1824

The University of Manchester

v6 signal processing extensions

- More support instructions
  - select (SEL)
    - chooses bytes in output according to corresponding GE flag
    - would follow (e.g.) SADD8
    - could be used for (e.g.) clipping samples
  - sum of differences

```
USAD8  Rd, Rm, Rs
```

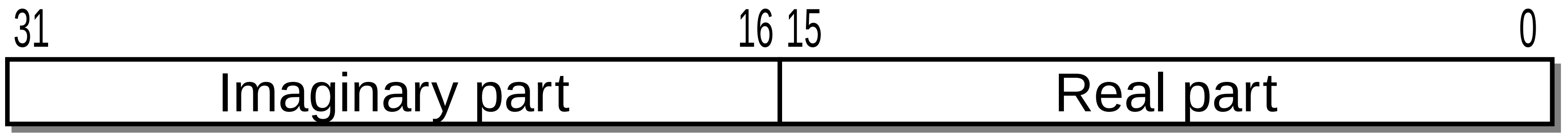
  - sum the absolute differences of the individual bytes in Rm, Rs
  - pattern matching (e.g. in MPEG encoding)
  - also available with accumulate

MANCHEStER
1824

The University
of Manchester

v6 signal processing extensions

- Example of use:
  - complex numbers packed into 32 bits



- Add
```
SADD16  R0, R1, R2
```
- Modulus
```
SMUAD   R0, R1, R1
```
- Multiply
```
SMUSD   R3, R1, R2    ; Real
SMUADX  R0, R1, R2    ; Imag.
PKHBT   R0, R3, R0, LSL #16
```

MANCHEStER
1824

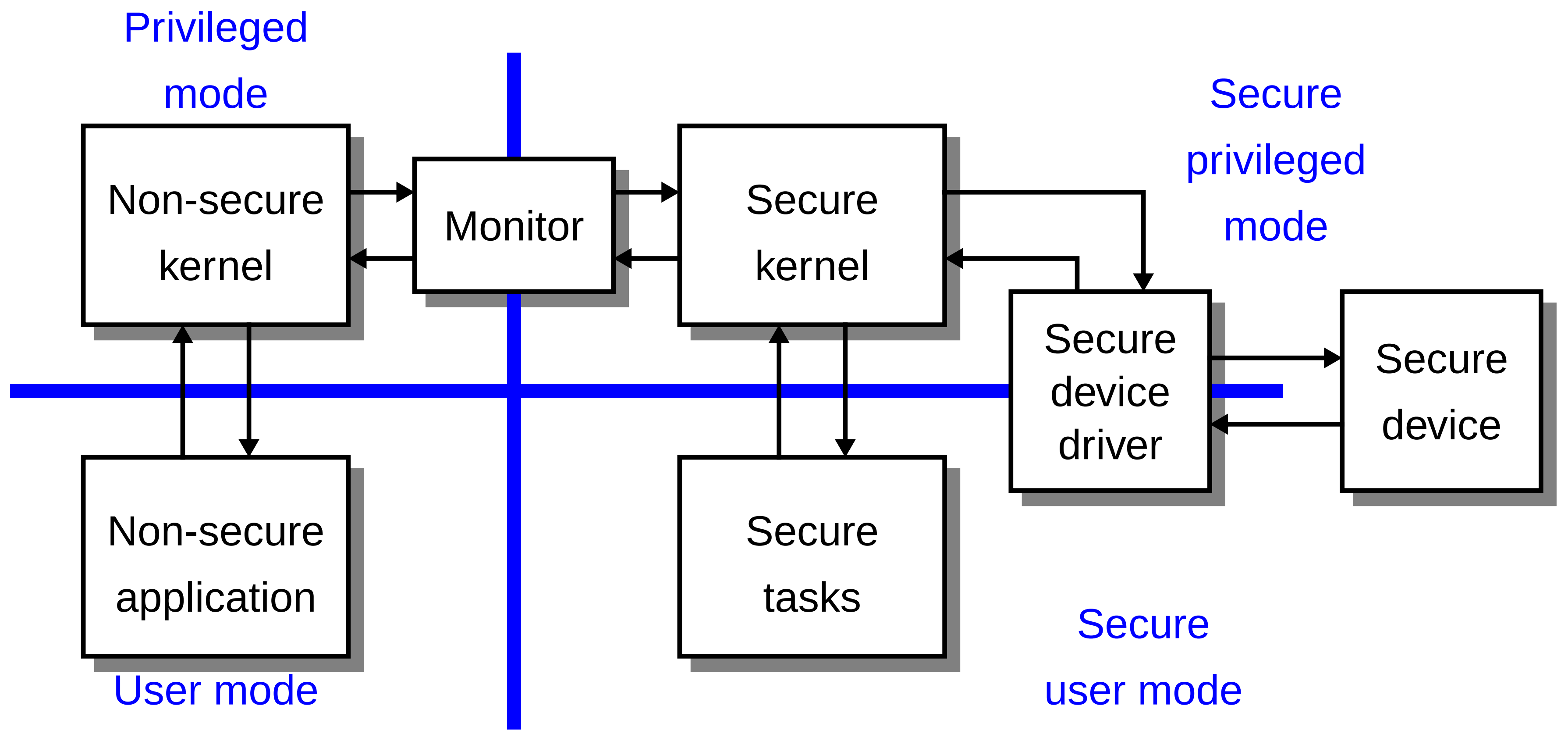
The University
of Manchester

Architectural extensions

- Outline:
  - instruction set extensions
  - digital signal processing instructions
  - ➜ **security extensions**
  - Java support
  - future instruction set developments

  ☞ hands-on: Thumb C and cycle counts

TrustZone™



- “NS” (Non-secure) bit determines the security status
  - held in system coprocessor
  - can only be changed via (trusted) secure monitor code

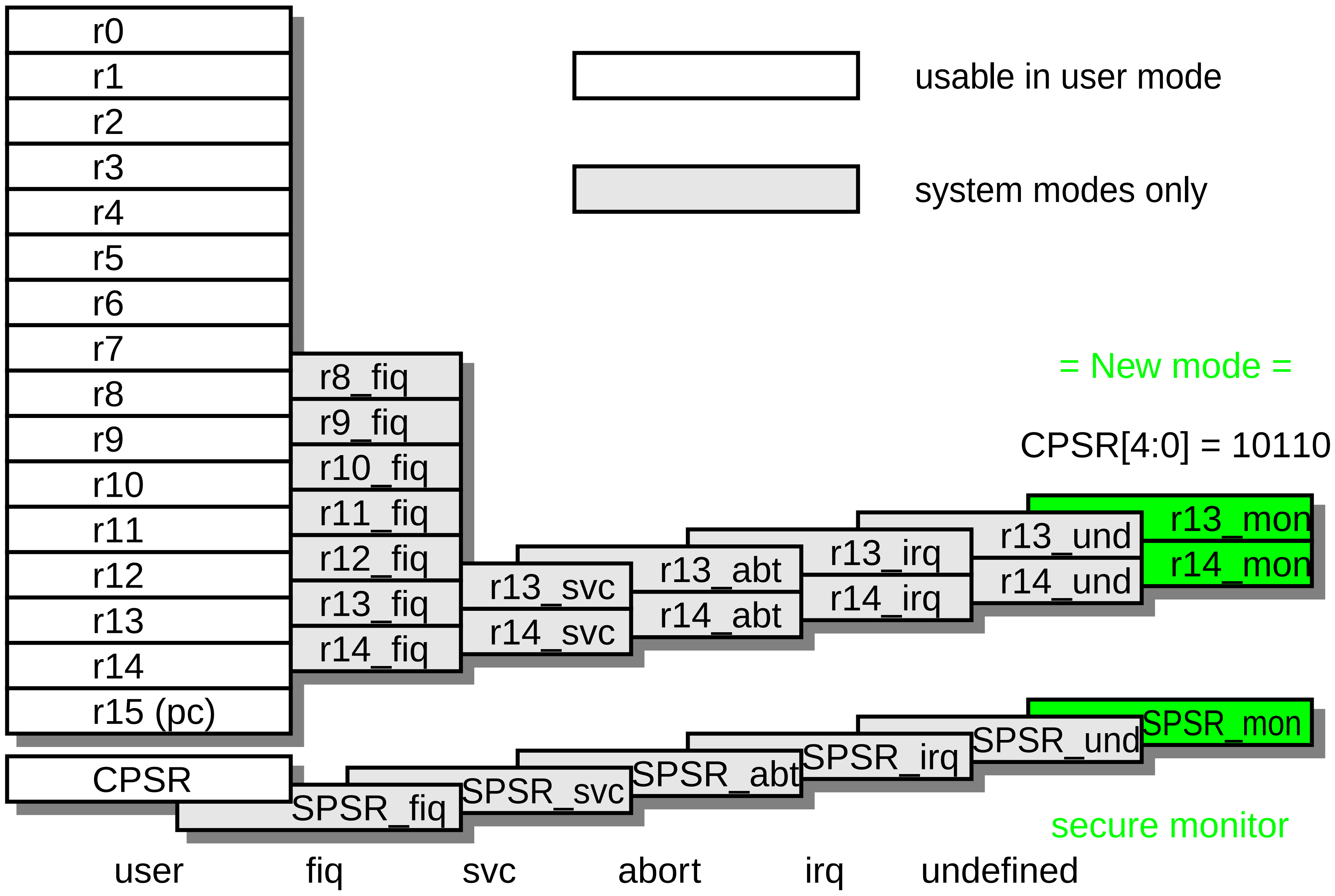
TrustZone™

- Secure monitor mode
  - processor operating mode – new to v6
  - privileged
  - *always* secure
  - entered via SMI (Software Monitor Instruction)
    - only works from privileged mode
    - causes undefined instruction exception from user mode
  - intended for switching security status
    - change NS bit
    - return

MANCHEstER 1824

The University of Manchester

TrustZone™



TrustZone™

- TrustZone affects the memory management (see later)
- Memory regions can be marked as:
  - Non-secure
    - always available
  - Secure
    - available only to ‘secure’ code
    - non-secure access attempt will abort
- Otherwise code is unaffected
  - reset $\Rightarrow$ secure mode

Architectural extensions

- Outline:
  - instruction set extensions
  - digital signal processing instructions
  - security extensions
  - ➜ **Java support**
  - future instruction set developments

  ☞ hands-on: Thumb C and cycle counts

Jazelle™

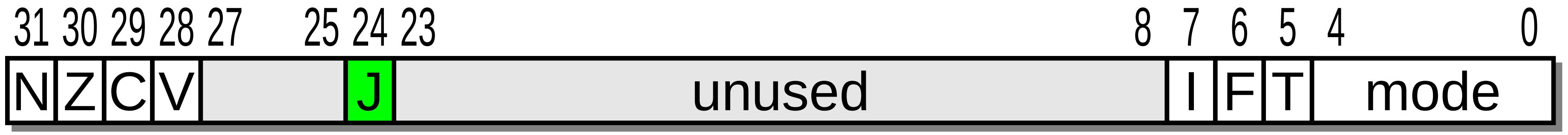
- Jazelle is a hardware instruction decoder
- Java byte codes are translated into ARM instructions
  - similar – in principle – to Thumb
  - translates *some* (140) Java byte codes
    - translation is *dynamic* (e.g. register specifiers are not fixed)
  - the codes processed account for most of the codes encountered in typical code
  - non-translated codes (94) trap for software emulation
  - performance is 8x that of software JVM

MANCHEStER 1824

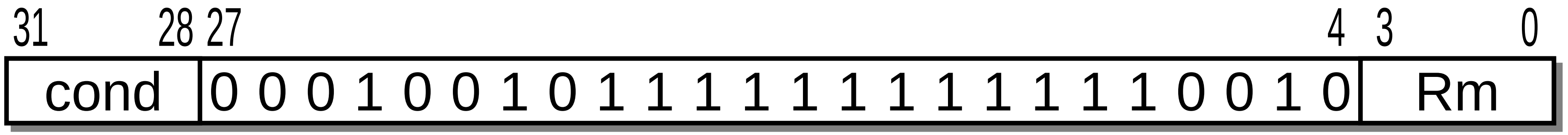
The University of Manchester

Jazelle™

- Jazelle mode indicated by a flag in CPSR



- Entered using `BXJ Rm` instruction



- Exception processing done in ARM code

Jazelle™ register use

- Many ARM registers have predefined functions in Jazelle

Register	Jazelle™ role
0-3	Cache of Java expression stack
4	Local variable 0 (‘this’ pointer)
5	Pointer to table of SW handlers
6	Java stack pointer
7	Java variables pointer
8	Java constant pool pointer
9-11	Reserved for JVM (no HW function)
12	Scratch reg.
13	Stack pointer
14	Link address / scratch register
15	Program counter

MANCHEstER
1824

The University
of Manchester

Architectural extensions

❑ Outline:

- ❍ instruction set extensions
- ❍ digital signal processing instructions
- ❍ security extensions
- ❍ Java support
- ➜ **future instruction set developments**

☞ hands-on: Thumb C and cycle counts

MANCHEStER 1824

The University of Manchester

Thumb 2

- Details not available
- Claims:
  - new instruction set
    - both 16- and 32-bit instructions
    - ARM-like instructions
      - some new operations {bitfield manipulation, jump tables, …}
  - ARM-like performance
  - Thumb-like code size

Hands-on: Thumb C and cycle counts

- See how to compile C programs into Thumb code
- Look at performance evaluation within the ARM software development tools
  - See how many clock cycles ARM and Thumb programs take

☞ Follow the 'Hands-on' instructions