Charm Reference Card

Registers: rd, rm, rn - can be r0, r1, … r15, pc, lr, sp

Immediates: imm_{12}, imm_{16}, imm_{20} - 12, 16, and 20 bit 2's complement

Load/Store Instructions

opcd: instruction	instruction semantics
0x1?: ldr rd, *mem*	loads word from *mem* into rd
0x2?: ldb rd, *mem*	loads byte from *mem* into rd
0x3?: str rd, *mem*	stores word from rd to *mem*
0x4?: stb rd, *mem*	stores byte from rd to *mem*

PC relative load/store, translated to 0x?2, where rm is 0xf

0x?9: !*label* Access 32-bit address at PC ± offset to *label*

opcd: addr mode	addr semantics	Instr Fmt
0x?0: $addr_{20}$	Access 20-bit unsigned $addr_{20}$	F2
0x?1: [rm]	Access 32-bit address in rm	F3
0x?2: [rm, imm_{16}]	Access 32-bit address in rm + imm_{16}	F3
0x?3: [rm, rn]	Access 32-bit address in rm + rn	F1
0x?4: [rm, imm_{16}]!	rm ← rm + imm_{16}, access 32-bit address in rm	F3
0x?5: [rm, rn]!	rm ← rm + rn bytes, access 32-bit address in rm	F1
0x?6: [rm], imm_{16}	Access 32-bit address in rm, rm ← rm + imm_{16}	F3
0x?7: [rm], rn	Access 32-bit address in rm, rm ← rm + rn	F1
0x?8: [rm, rn, imm_{12}]	rn shifts imm_{12}, Access 32-bit address in rm + rn	F1A

Arithmetic and Logic Instructions

opcd: instruction	instruction semantics
0x?0: add rd, rm, op2	rd ← rm + op2 integer add
0x?1: sub rd, rm, op2	rd ← rm - op2 integer subtract
0x?2: mul rd, rm, op2	rd ← rm * op2 integer multiply
0x?3: div rd, rm, op2	rd ← rm / op2 integer divide
0x?4: mod rd, rm, op2	rd ← rm % op2 integer modulo
0x?5: and rd, rm, op2	rd ← rm & op2 bitwise and
0x?6: orr rd, rm, op2	rd ← rm | op2 bitwise or
0x?7: eor rd, rn, op2	rd ← rn ^ op2 bitwise xor
0x?8: adc rd, rm, op2	rd ← rm + op2 integer add with carry
0x?9: sbc rd, rm, op2	rd ← rm - op2 integer subtract with carry
0x?a: adf rd, rm, op2	rd ← rm + op2 floating point add
0x?b: suf rd, rm, op2	rd ← rm - op2 floating point subtract
0x?c: muf rd, rm, op2	rd ← rm * op2 floating point multiply
0x?d: dif rd, rm, op2	rd ← rm / op2 floating point divide

op2 can be

opcd: op2 value	op2 semantics	Instr Fmt
0x5?: rm	Value in register rm	F1
0x6?: imm_{16}	16-bit immediate imm_{16}	F3

Charm Code	C Code
0x200 ldr r0, 0x100	int x is at address 0x100
0x204 cmp r0, 0	if (x)
0x208 beq 0x214	x = x + 1;
0x20c add r0, r0, 1	else
0x210 bal 0x218	x = x + 2;
0x214 add r0, r0, 2	
0x218 str r0, 0x100	

Move, Compare, Shift Instructions

opcd: instruction	instruction semantics
0x?0: mov rd, op2	rd ← op2 op2 is 2's compl
0x?1: mva rd, op2	rd ← op2 op2 is a pos num
0x?2: cmp rd, op2	set N and Z flags in CPSR for rd - op2
0x?3: tst rd, op2	set N and Z flags in CPSR for rd & op2
0x?4: teq rd, op2	set N and Z flags in CPSR for rd ^ op2
0x?5: shf rd, op2	shift rd logical by op2%32. op2>0 shifts left, op2<0 shifts right
0x?6: sha rd, op2	shift rd arithmetic by op2%32. op2>0 shifts left, op2<0 shifts right sha - left shift propagates bit 0, right shift bit 31
0x?7: rot rd, op2	rotate rd by op2. op2%32 is shift value, op2>0 rotates left, op2<0 rotates right
0x?8: one rd, op2	Create one's complement of op2 and place it in rd.
0x?9: fti rd, op2	Convert floating point op2 to integer in rd.
0x?a: itf rd, op2	Convert integer op2 to floating point in rd.
0x?b: cmf rd, op2	Compare floating point values.

op2 can be

opcd: op2 value	op2 semantics	Instr Fmt
0x7?: rm	Value in register rm	F1
0x8?: imm_{20}	20-bit 2's complement number in imm_{20}	F2

Charm Reference Card

Registers: `rd`, `rm`, `rn` - can be `r0`, `r1`, … `r15`, `pc`, `lr`, `sp` **Immediates:** imm_{12}, imm_{16}, imm_{20} - 12, 16, and 20 bit 2's complement

Branch Instructions

opcd: instruction	instruction semantics
`0x?0: bal dest`	set `pc` to `dest`
`0x?1: beq dest`	set `pc` to `dest` if `Z == 1`
`0x?2: bne dest`	set `pc` to `dest` if `Z == 0`
`0x?3: blt dest`	set `pc` to `dest` if `N != V`
`0x?4: ble dest`	set `pc` to `dest` if `(Z == 1) or (N != V)`
`0x?5: bgt dest`	set `pc` to `dest` if `(Z == 0) and (N == V)`
`0x?6: bge dest`	set `pc` to `dest` if `N == V`
`0x?7: blr dest`	set `lr` to `pc+4`, set `pc` to `dest`

`dest` can be

opcd: `dest` value		`dest` value semantics	Instr Fmt
`0x9?:`	label or number	20-bit address $addr_{20}$ is not 2's complement	F2
`0xa?:`	`[rd]`	32-bit address in register `rd`	F2
`0xb?:`	!label	32-bit pc relative address, imm_{20} added to `rd` that is 15	F2
		!label is assembly notation and not strictly ISA	

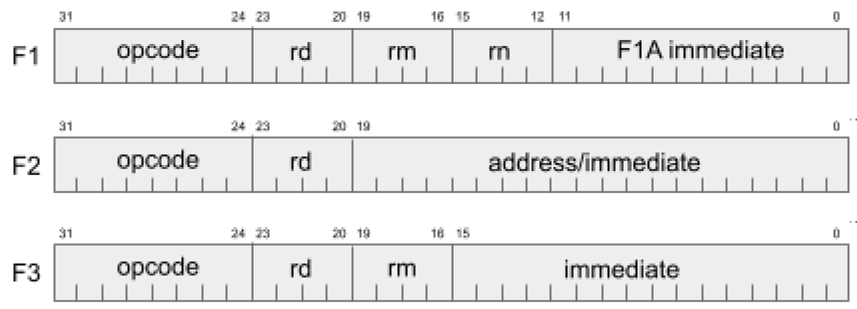
Kernel Mode Instructions

			Instr Fmt
`0xc0: ker` imm_{20}	`r0`←imm_{20}, `r14←return addr, r15←`OS base addr, U		F2
`0xc1: srg` imm_{20}	bit 5 - `1/0` set/clear bit in `cpsr`, `bits 0-4` - bit pos to set/clear		F2
`0xc2: ioi` imm_{20}	perform I/O at I/O location imm_{20}, imm_{20} determines I/O	`0xc3: rfi` imm_{20} imm_{20}`=0/1`, return from ker/tmr mode	F2
`0xc4: mkd krd, rm`	`krd ← rm` Move between regular regs and kernel regs. mkd - kreg is dest, mks - kreg is source.	`mkd r6, r14 ir14 ← r14`	F1
`0xc5: mks rd, krm`	`rd ← krm` Kernel Regs: kr0-kr15, kr0:cpsr, kr1:kpsr, kr2:kr13, kr3:kr14, kr4:irsr, kr5:ir13, kr6:ir14	`mks r14, r10 r14 ← kr10`	F1

Charm Register and Calling Conventions

`r0` - **`r3`** - Scratch registers. Up to 4 params are passed in them. **`r0`** - Function return value, **`r4`** - **`r12`** - General registers. If a function uses these, they must be saved and restored **`r13`** - Stack Pointer, also referred to as **`sp`**, **`r14`** - Link register, also referred to as **`lr`**, **`r15`** - Program counter, also referred to as **`pc`**

Charm Instruction Formats

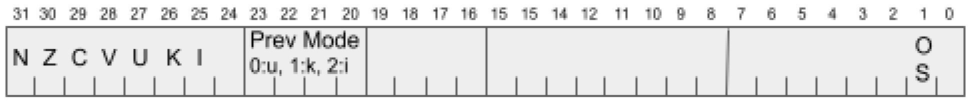


`add rd, rm, rn`
`ldr rd, [rm], rn`
`mov rd, rm`
`ldr rd, [rm, rn,` imm_{12}`] F1A`

`ldr rd, addr`
`mov rd,` imm_{20}
`mva rd, addr`

`ldr rd, [rm,` imm_{16}`]`
`ldr rd, [rm],` imm_{16}
`add rd, rm,` imm_{16}

Charm Status Register (CPSR)



N, Z - Negative and zero flags, cmp and beq, C - Carry flag, V - Overflow flag, U, K, I - User mode, kernel mode, and Interrupt mode flags, OS - OS loaded flag