

COMMON MIDTERM MISTAKES

1. Only LDR and STR can access memory:

int32_t a32, b32, c32 ;	LDR R0,b32 // R0 ← copy of b32
a32 = b32 + c32 ;	ADD R0,R0,c32 // ADD can't access memory
	LDR R1,c32 // R1 ← copy of c32
	ADD R0,R0,R1 // R0 ← R0 + R1
	STR R0,a32 // R0 → a32

2. Translated C assignment statements require an STR at the end:

int32_t a32, b32 ;	LDR R0,a32 // R0 ← copy of a32
a32 = a32 + b32 ;	LDR R1,b32 // R1 ← copy of b32
	ADD R0,R0,R1 // R0 ← R0 + R1
	STR R0,a32 // R0 → a32

3. Constants are only allowed as the Last operand of ADD or SUB:

int32_t a32, b32 ;	LDR R0,b32 // R0 ← copy of b32
a32 = 5 - b32 ;	SUB R0,5,R0 // 2nd operand must be reg
	RSB R0,R0,5 // R0 ← 5 - R0
	STR R0,a32 // R0 → a32

4. Don't copy variable into a register if not used to compute the result:

int32_t a32, b32, c32 ;	LDR R0,a32 // a32 is output only!!
a32 = b32 + c32 ;	LDR R1,b32 // R1 ← copy of b32
	LDR R2,c32 // R2 ← copy of c32
	ADD R0,R1,R2 // R0 ← R1 + R2
	STR R0,a32 // R0 → a32

5. Addition/Subtraction of 64-bit operands requires a two-instruction sequence:

int64_t a64, b64 ;	LDRD R0,R1,b64 // R1.R0 ← copy of b64
a64 = b64 + 5 ;	ADD R0,R0,5 // Only adds lower 32 bits
	ADDs R0,R0,5 // R0 ← R0 + 5 (save carry)
	ADC R1,R1,0 // R1 ← R1 + 0 + carry
	STRD R0,R1,a64 // R1.R0 → a64

6. ALL operands of all multiplies and divides must be registers:

int32_t a32, b32 ;	LDR R0,b32 // R0 ← copy of b32
a32 = 5 * b32 ;	MUL R0,R0,5 // All operands must be regs
	LDR R1,=5 // R1 ← 5
	MUL R0,R0,R1 // R0 ← R0 * R1
	STR R0,a32 // R0 → a32

7. Pay attention to signed vs. unsigned and size:

int8_t s8 ;	int16_t s16 ;	uint8_t u8 ;	uint16_t u16 ;
int32_t s32 ;	int64_t s64 ;	uint32_t u32 ;	uint64_t u64 ;
s8 = 0 ;	// use STRB(same)	u8 = 0 ;	// use STRB(same)
s16 = 0 ;	// use STRH(same)	u16 = 0 ;	// use STRH(same)
s32 = (int32_t) s8 ;	// use LDRSB	u32 = (int32_t) u8 ;	// use LDRB
s32 = (int32_t) s16 ;	// use LDRSH	u32 = (int32_t) u16 ;	// use LDRH
s32 = 5 * s32 ;	// use MUL (same)	u32 = 5 * u32 ;	// use MUL (same)
s32 = s32 / 5 ;	// use SDIV	u32 = u32 / 5 ;	// use UDIV
s64 = (int64_t) s32 ;	// sign extend	u64 = (uint64_t) u32 ;	// zero extend

8. Functions require a return instruction:

void Dumb(void) { return ; }	Zero: BX LR // Return to where called
---------------------------------------	---------------------------------------

COMMON MIDTERM MISTAKES

9. Functions return result in R0:

<pre>int32_t Zero(void) { return 0 ; }</pre>	<pre>Zero: LDR R0,=0 // R0 ← 0 STR R0,Zero // "Zero" is not a variable! BX LR // Returned value is in R0</pre>
--	--

10. Function parameters cannot be accessed using C identifiers:

<pre>void Add1(int32_t a32) { return a + 1 ; }</pre>	<pre>Add1: LDR R0,a32 // a32 already in R0!! ADD R0,R0,1 // R0 ← R0 + 1 BX LR // return to where called</pre>
--	---

11. Calling a function changes the contents of LR and possibly R0-R3 and R12:

<pre>void foo(int32_t a32) { return a32 + bar(0) ; }</pre>	<pre>foo: PUSH {R4,LR} // preserve R4 and LR MOV R4,R0 // R4 is safe place for a32 LDR R0,=0 // R0 ← 0 (param for bar) BL bar // changes R0-R3 and R12! ADD R0,R0,R4 // R0 ← bar(0) + a32 POP {R4,LR} // restore R4 and LR BX LR // return to where call</pre>
--	--

12. Subscript of array reference must be multiplied by operand size:

<pre>int8_t a8[10] ; int32_t k32 ; a8[k32] = 0 ;</pre>	<pre>LDR R0,=0 // R0 ← 0 ADR R1,a8 // R1 ← &a8[0] LDR R2,k32 // R2 ← copy of k32 STRB R0,[R1,R2] // adr = R1 + R2</pre>
<pre>int16_t a16[10] ; int32_t k32 ; a16[k32] = 0 ;</pre>	<pre>LDR R0,=0 // R0 ← 0 ADR R1,a16 // R1 ← &a16[0] LDR R2,k32 // R2 ← copy of k32 STRH R0,[R1,R2,LSL 1] // adr = R1 + 2*R2</pre>
<pre>int32_t a32[10] ; int32_t k32 ; a32[k32] = 0 ;</pre>	<pre>LDR R0,=0 // R0 ← 0 ADR R1,a32 // R1 ← &a32[0] LDR R2,k32 // R2 ← copy of k32 STR R0,[R1,R2] // Array of bytes only STR R0,[R1,R2,LSL 2] // adr = R1 + 4*R2</pre>

13. Use ADR with an array, LDR with a pointer:

<pre>int8_t a8[10] ; int32_t k32 ; a8[k32] = 0 ;</pre>	<pre>LDR R0,=0 // R0 ← 0 ADR R1,a8 // R1 ← &a8[0] LDR R2,k32 // R2 ← copy of k32 STRB R0,[R1,R2] // adr = R1 + R2</pre>
<pre>int8_t *p8 ; int32_t k32 ; *(p8 + k32) = 0 ;</pre>	<pre>LDR R0,=0 // R0 ← 0 LDR R1,p8 // R1 ← copy of p8 LDR R2,k32 // R2 ← copy of k32 STRB R0,[R1,R2] // adr = R1 + R2</pre>

14. If-Then-Else requires an unconditional branch:

<pre>int32_t a32, b32, c32 ; if (a32 == 0) b32 = 0 ; else c32 = 1 ;</pre>	<pre>LDR R0,a32 // R0 ← copy of a32 CMP R0,0 // Is R0 == 0? BNE Else // if not, goto else Then: LDR R0,=0 // Then: R0 ← 0 STR R0,b32 // R0 → b32 B Done // skip over else!! Else: LDR R0,=1 // Else: R0 ← 1 STR R0,c32 // R0 → c32 Done: ... // (next instruction)</pre>
--	--