Accessing memory

P. Bernardi, R. Ferrero

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

Memory map

Vendor specific	0xFFFFFFFF 0xE0100000	≈ 0.5 GB
Private peripheral bus: debug/external	0xE00FFFFF 0xE0040000	1 MB, 256 kB
Private peripheral bus: internal	0xE003FFFF 0xE0000000	256 kB
External device	0xDFFFFFFF 0xA0000000	1 GB
External RAM	0x9FFFFFFF 0x60000000	1 GB
Peripherals	0x5FFFFFFF 0x40000000	0.5 GB
SRAM	0x3FFFFFFF 0x20000000	0.5 GB
Code	0x1FFFFFFF 0x00000000	0.5 GB

Memory access attributes

Region	Bufferable	Cacheable	Executable
Code	yes	write through	yes
SRAM	yes	write back	yes
Peripherals	no	no	no
External RAM	no	write back	yes
External device	no	no	no
System	no	no	no

Load and store pseudo-intructions

`load/store <Rd>, <addressing_mode>`

Load	Store	Size and type
LDR	STR	word (32 bits)
LDRB	STRB	byte (8 bits)
LDRH	STRH	halfword (16 bits)
LDRSB	–	signed byte
LDRSH	–	signed halfword
LDRD	STRD	two words
LDM	STM	multiple words

Exercise

If `r0` = 0x00008004, what are the values of the registers `r1`-`r5` after the following instructions?

```
LDR r1, [r0]
LDRB r2, [r0]
LDRH r3, [r0]
LDRSB r4, [r0]
LDRSH r5, [r0]
```

0x41	0x00008000
0x73	0x00008001
0x73	0x00008002
0x65	0x00008003
0x8D	0x00008004
0x62	0x00008005
0x6C	0x00008006
0x79	0x00008007

Exercise

If `r0` = 0x00008004, what a… the registers `r1`-`r5` after the following instructions?

r1 = 0x796C628D
r2 = 0x0000008D
r3 = 0x0000628D
r4 = 0xFFFFFF8D
r5 = 0x0000628D

```
LDR r1, [r0]
LDRB r2, [r0]
LDRH r3, [r0]
LDRSB r4, [r0]
LDRSH r5, [r0]
```

0x41	0x00008000
0x73	0x00008001
0x73	0x00008002
0x65	0x00008003
0x8D	0x00008004
0x62	0x00008005
0x6C	0x00008006
0x79	0x00008007

LDRD

`LDRD <Rd1>, <Rd2>,<addressing_mode>`

- It loads two registers
- Example:

`LDRD r1, r2,[r0]`

if `r0` = 0x00008000 then

`r1` = 0x65737341

`r2` = 0x796C628D

0x41	0x00008000
0x73	0x00008001
0x73	0x00008002
0x65	0x00008003
0x8D	0x00008004
0x62	0x00008005
0x6C	0x00008006
0x79	0x00008007

STR

- It copies the content of a register into four consecutive memory locations.
- Example:

`STR r1, [r0]`

`r0` = 0x20000000

`r1` = 0x65737341

`r2` = 0x796C628D

0x41	0x20000000
0x73	0x20000001
0x73	0x20000002
0x65	0x20000003
0x00	0x20000004
0x00	0x20000005
0x00	0x20000006
0x00	0x20000007

STRB

- It copies the least significant byte (LSB) of a register into the memory location.
- Example:

```
STRB r1, [r0]
```

`r0` = 0x20000000

`r1` = 0x65737341

`r2` = 0x796C628D

0x41	0x20000000
0x00	0x20000001
0x00	0x20000002
0x00	0x20000003
0x00	0x20000004
0x00	0x20000005
0x00	0x20000006
0x00	0x20000007

STRH

- It copies the lower 16-bit content of a register into two consecutive memory locations.
- Example:

```
STRH r1, [r0]
```

`r0` = 0x20000000

`r1` = 0x65737341

`r2` = 0x796C628D

0x41	0x20000000
0x73	0x20000001
0x00	0x20000002
0x00	0x20000003
0x00	0x20000004
0x00	0x20000005
0x00	0x20000006
0x00	0x20000007

STRD

- It copies the content of two registers into eight consecutive memory locations.
- Example:

`STR r1, r2, [r0]`

`r0` = 0x20000000

`r1` = 0x65737341

`r2` = 0x796C628D

0x41	0x20000000
0x73	0x20000001
0x73	0x20000002
0x65	0x20000003
0x8D	0x20000004
0x62	0x20000005
0x6C	0x20000006
0x79	0x20000007

Addressing mode

- Addressing:
  - pre-indexed
    - with writeback
    - without writeback
  - post-indexed
- Offset:
  - fixed value
  - shifted register

Pre-indexed addressing

- The address is computed by summing the offset to the value in the base register `Rn`:

```
load/store <Rd>, [<Rn>,<offset>]{!}
```

- the offset is either a 12-bit constant or a register, which can be shifted left up to 3 positions.
- `!` is optional and indicates if `Rn` is updated at the end of the instruction (only with constant offset).

Pre-indexed addressing: example

- Using pre-indexed addressing, write the instructions for loading 4 words from memory into registers `r2`-`r5`.
- Register `r0` contains the address of the first byte of the block of memory.
  e.g., `r0` = 0x00008000.

With constant offset

```
LDR r2, [r0]
LDR r3, [r0, #4]
LDR r4, [r0, #8]
LDR r5, [r0, #12]
```

At the end, `r0` = 0x00008000

With constant offset and writeback

```
LDR r2, [r0]
LDR r3, [r0, #4]!
LDR r4, [r0, #4]!
LDR r5, [r0, #4]!
```

At the end, `r0` = 0x0000800C

With register as offset

```
LDR r2, [r0]
MOV r1, #4
LDR r3, [r0, r1]
MOV r1, #8
LDR r4, [r0, r1]
MOV r1, #12
LDR r5, [r0, r1]
```

With shifted register as offset

```
LDR r2, [r0]
MOV r1, #4
LDR r3, [r0, r1]
LDR r4, [r0, r1, LSL #1]
MOV r1, #12
LDR r5, [r0, r1]
```

Post-indexed addressing

- The address is given by the base register `Rn`:

  `load/store <Rd>, [<Rn>], <offset>`
- Then `Rn` is updated by adding the offset.
- the offset can be a 12-bit constant or a register, which can be shifted left up to 3 positions.
- `!` is missing because `Rn` is always updated.

Look-up table

- A look-up table is an array of pre-calculated constants.
- Pro: frequently used values are not computed at run-time or are computed only the first time
- Con: additional memory space is required.
- The look-up table can be easily accessed with indexed addressing

Example: look-up table of bytes

- Write a program that uses the x value contained in `r2` and sets `r4` with the value of $x^2 + 2x + 1$.
- Assume $0 \leq x \leq 10$.

Example: look-up table of bytes

```
    AREA    |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    MOV r2, #8    ;after some calculus
    LDR r0, =lookup
    LDRB r4, [r0, r2]
stop B stop        ;stop program
lookup DCB 1, 4, 9, 16, 25, 36, 49,
64, 81, 100
    ENDP
```

Example: look-up table of words

- Write a program that uses the x value contained in `r2` and sets `r4` with the factorial of x.
- Assume $0 \leq x \leq 10$.

Example: look-up table of words

```
    AREA    |.text|, CODE, READONLY
Reset_Handler   PROC
    EXPORT  Reset_Handler [WEAK]
    MOV r2, #8   ;after some calculus
    LDR r0, =lookup
    LDR r4, [r0, r2, LSL #2]
stop B stop         ;stop program
lookup DCD 1, 1, 2, 6, 24, 120, 720,
5040, 40320, 362880, 3628800
    ENDP
```