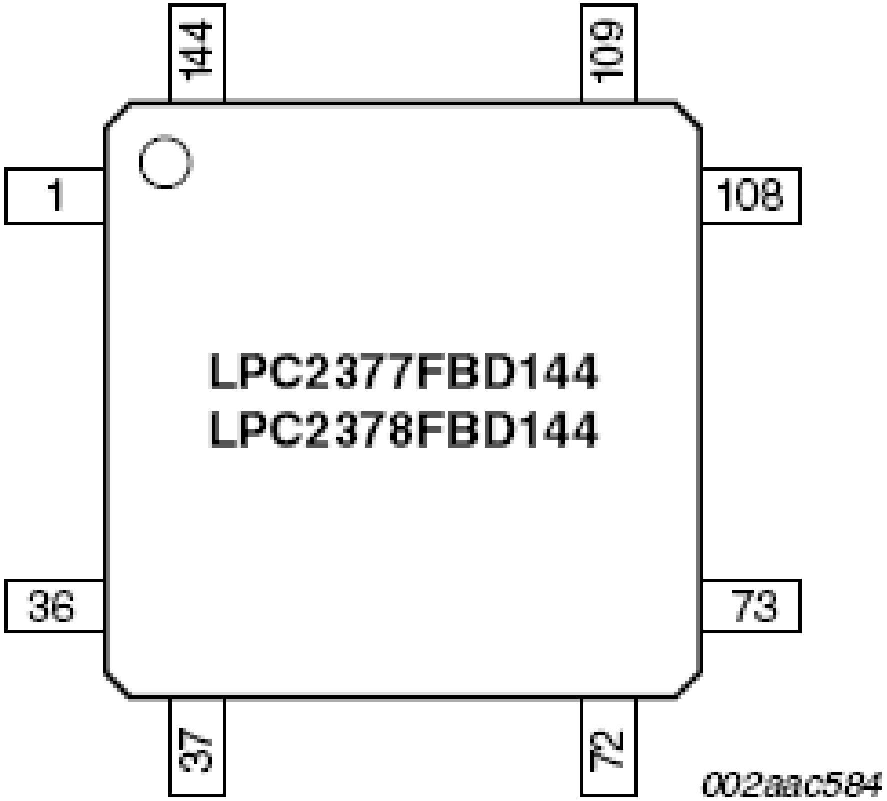
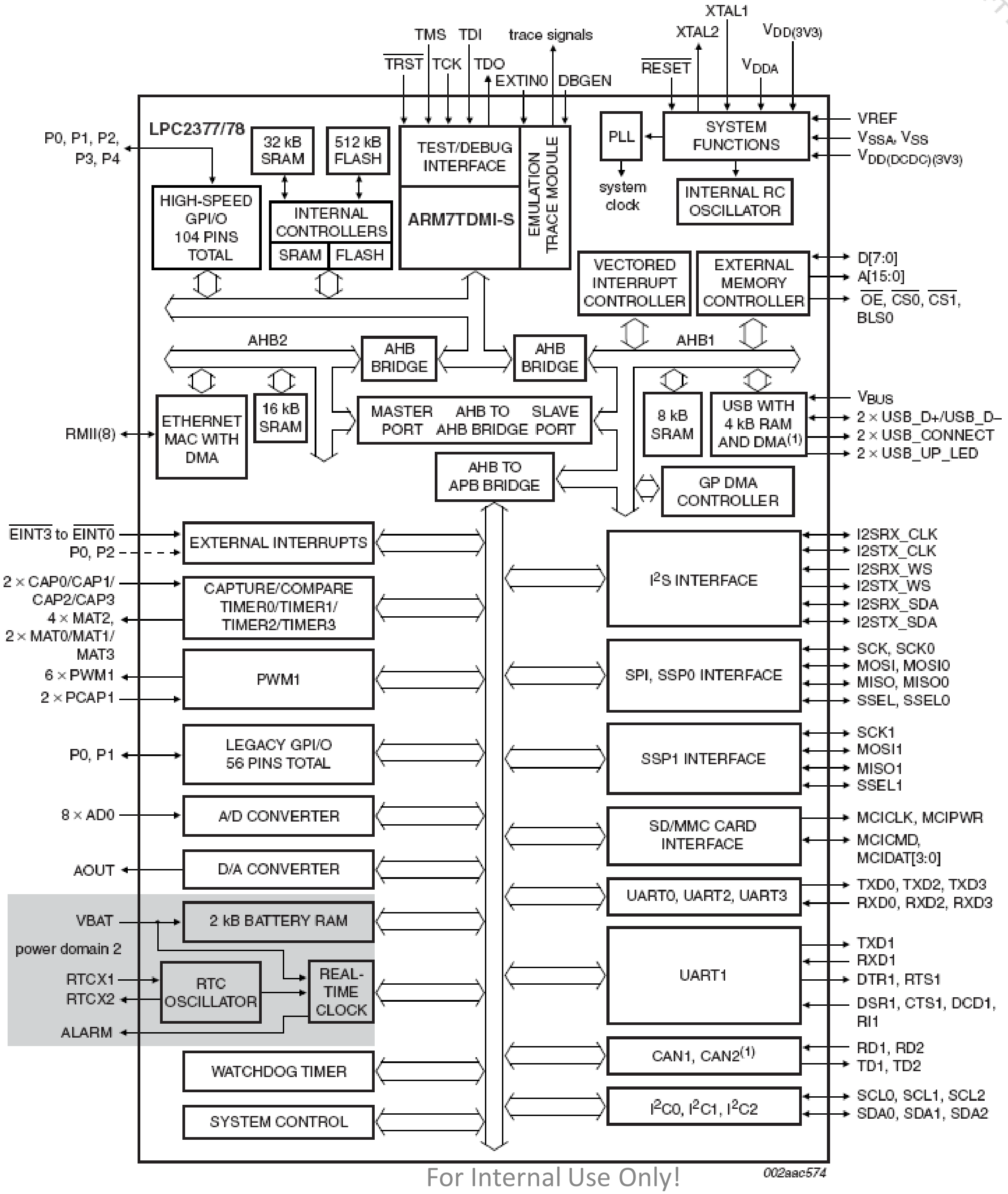
Embedded Systems and Microprocessor Systems

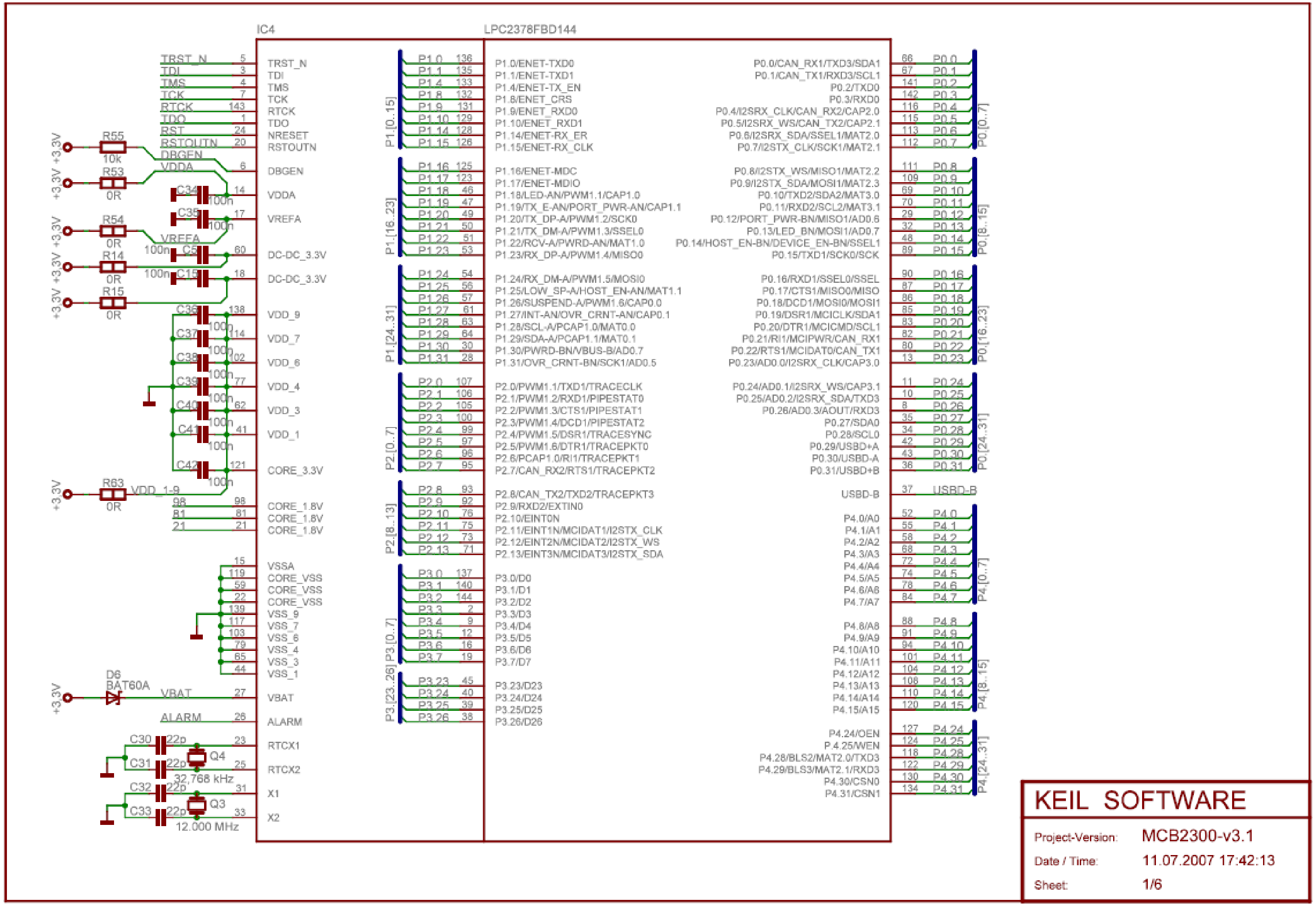
review

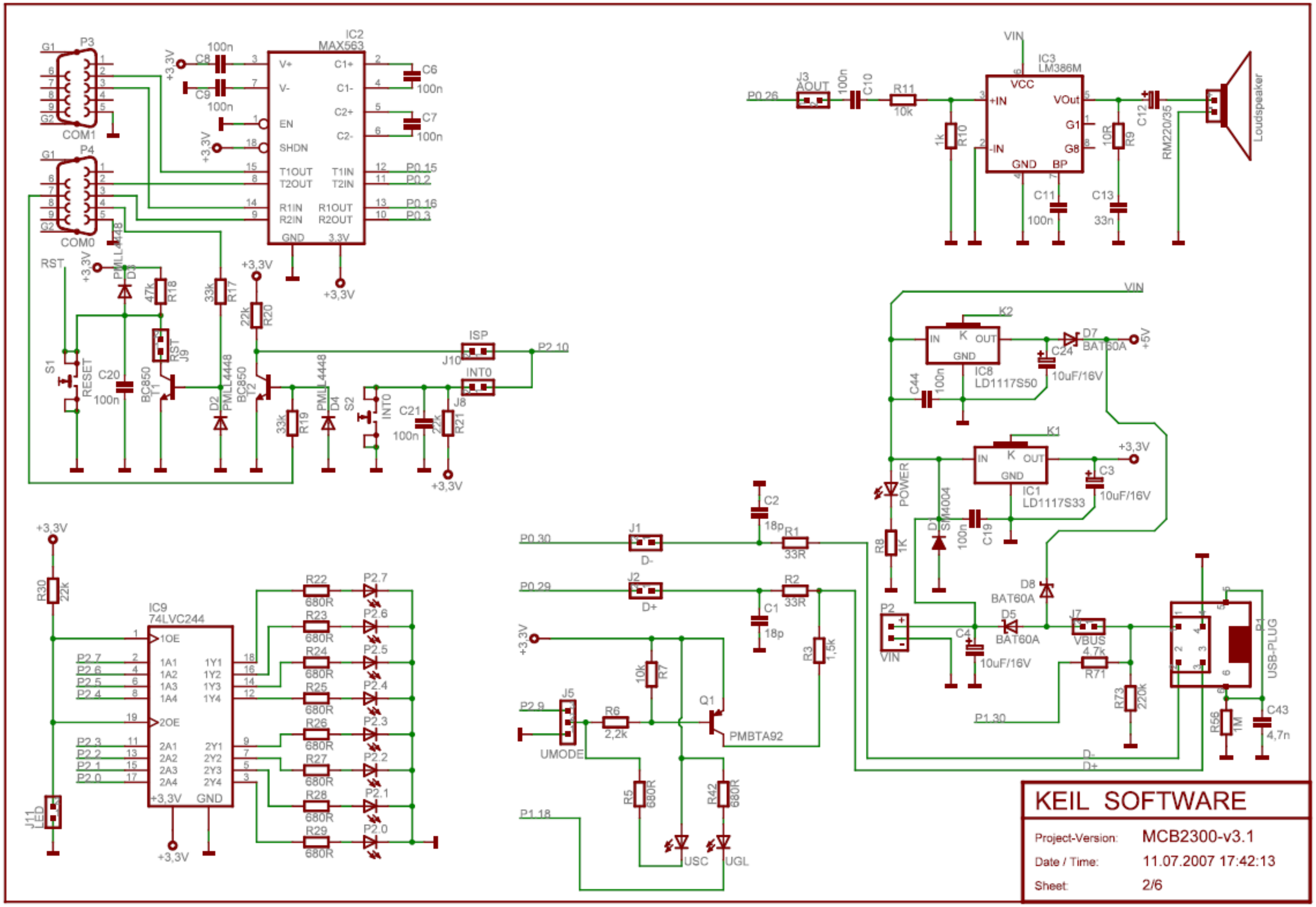
- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

- ✓ **Organization/Architecture**
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes


144
109
1
108
LPC2377FBD144
LPC2378FBD144
36
73
37
72
002aac584


LPC2377/78
P0, P1, P2, P3, P4
HIGH-SPEED GPI/O 104 PINS TOTAL
32 kB SRAM
512 kB FLASH
INTERNAL CONTROLLERS
SRAM
FLASH
TMS
TDI
trace signals
TRST
TCK
TDO
EXTIN0
DBGEN
TEST/DEBUG INTERFACE
ARM7TDMI-S
EMULATION TRACE MODULE
XTAL1
XTAL2
RESET
VDD(3V3)
VDDA
PLL
system clock
SYSTEM FUNCTIONS
INTERNAL RC OSCILLATOR
VREF
VSSA, VSS
VDD(DCDC)(3V3)
VECTORED INTERRUPT CONTROLLER
EXTERNAL MEMORY CONTROLLER
D[7:0]
A[15:0]
OE, CS0, CS1, BLS0
AHB2
AHB BRIDGE
AHB BRIDGE
AHB1
RMII(8)
ETHERNET MAC WITH DMA
16 kB SRAM
MASTER PORT
AHB TO AHB BRIDGE
SLAVE PORT
8 kB SRAM
USB WITH 4 kB RAM AND DMA(1)
VBUS
2 × USB_D+/USB_D−
2 × USB_CONNECT
2 × USB_UP_LED
AHB TO APB BRIDGE
GP DMA CONTROLLER
EINT3 to EINT0
P0, P2
EXTERNAL INTERRUPTS
2 × CAP0/CAP1/ CAP2/CAP3
4 × MAT2,
2 × MAT0/MAT1/ MAT3
CAPTURE/COMPARE TIMER0/TIMER1/ TIMER2/TIMER3
6 × PWM1
2 × PCAP1
PWM1
P0, P1
LEGACY GPI/O 56 PINS TOTAL
8 × AD0
A/D CONVERTER
AOUT
D/A CONVERTER
VBAT
power domain 2
2 kB BATTERY RAM
RTCX1
RTCX2
RTC OSCILLATOR
REAL-TIME CLOCK
ALARM
WATCHDOG TIMER
SYSTEM CONTROL
I2S INTERFACE
I2SRX_CLK
I2STX_CLK
I2SRX_WS
I2STX_WS
I2SRX_SDA
I2STX_SDA
SPI, SSP0 INTERFACE
SCK, SCK0
MOSI, MOSI0
MISO, MISO0
SSEL, SSEL0
SSP1 INTERFACE
SCK1
MOSI1
MISO1
SSEL1
SD/MMC CARD INTERFACE
MCICLK, MCIPWR
MCICMD, MCIDAT[3:0]
UART0, UART2, UART3
TXD0, TXD2, TXD3
RXD0, RXD2, RXD3
UART1
TXD1
RXD1
DTR1, RTS1
DSR1, CTS1, DCD1, RI1
CAN1, CAN2(1)
RD1, RD2
TD1, TD2
I2C0, I2C1, I2C2
SCL0, SCL1, SCL2
SDA0, SDA1, SDA2
002aac574


IC4
LPC2378FBD144
TRST_N
TDI
TMS
TCK
RTCK
TDO
NRESET
RSTOUTN
DBGEN
VDDA
VREFA
DC-DC_3.3V
DC-DC_3.3V
VDD_9
VDD_7
VDD_6
VDD_4
VDD_3
VDD_1
CORE_3.3V
CORE_1.8V
CORE_1.8V
CORE_1.8V
VSSA
CORE_VSS
CORE_VSS
CORE_VSS
VSS_9
VSS_7
VSS_6
VSS_4
VSS_3
VSS_1
VBAT
ALARM
RTCX1
RTCX2
X1
X2
R55
10k
R53
0R
R54
0R
R14
0R
R15
0R
R63
0R
+3.3V
C34
100n
C35
100n
C5
100n
C15
100n
C36
C37
C38
C39
C40
C41
C42
100n
RST
RSTOUTN
VDDA
VREFA
VDD_1-9
D6
BAT60A
VBAT
ALARM
C30
22p
C31
22p
Q4
32,768 kHz
C32
22p
C33
22p
Q3
12.000 MHz
P1.[0..15]
P1.[16..23]
P1.[24..31]
P2.[0..7]
P2.[8..13]
P3.[0..7]
P3.[23..26]
P1.0/ENET-TXD0
P1.1/ENET-TXD1
P1.4/ENET-TX_EN
P1.8/ENET_CRS
P1.9/ENET_RXD0
P1.10/ENET_RXD1
P1.14/ENET-RX_ER
P1.15/ENET-RX_CLK
P1.16/ENET-MDC
P1.17/ENET-MDIO
P1.18/LED-A/PWM1.1/CAP1.0
P1.19/TX_E-AN/PORT_PWR-AN/CAP1.1
P1.20/TX_DP-A/PWM1.2/SCK0
P1.21/TX_DM-A/PWM1.3/SSEL0
P1.22/RCV-A/PWRD-AN/MAT1.0
P1.23/RX_DP-A/PWM1.4/MISO0
P1.24/RX_DM-A/PWM1.5/MOSI0
P1.25/LOW_SP-A/HOST_EN-AN/MAT1.1
P1.26/SUSPEND-A/PWM1.6/CAP0.0
P1.27/INT-AN/OVR_CRNT-AN/CAP0.1
P1.28/SCL-A/PCAP1.0/MAT0.0
P1.29/SDA-A/PCAP1.1/MAT0.1
P1.30/PWRD-BN/VBUS-B/AD0.7
P1.31/OVR_CRNT-BN/SCK1/AD0.5
P2.0/PWM1.1/TXD1/TRACECLK
P2.1/PWM1.2/RXD1/PIPESTAT0
P2.2/PWM1.3/CTS1/PIPESTAT1
P2.3/PWM1.4/DCD1/PIPESTAT2
P2.4/PWM1.5/DSR1/TRACESYNC
P2.5/PWM1.6/DTR1/TRACEPKT0
P2.6/PCAP1.0/RI1/TRACEPKT1
P2.7/CAN_RX2/RTS1/TRACEPKT2
P2.8/CAN_TX2/TXD2/TRACEPKT3
P2.9/RXD2/EXTIN0
P2.10/EINT0N
P2.11/EINT1N/MCIDAT1/I2STX_CLK
P2.12/EINT2N/MCIDAT2/I2STX_WS
P2.13/EINT3N/MCIDAT3/I2STX_SDA
P3.0/D0
P3.1/D1
P3.2/D2
P3.3/D3
P3.4/D4
P3.5/D5
P3.6/D6
P3.7/D7
P3.23/D23
P3.24/D24
P3.25/D25
P3.26/D26
P0.0/CAN_RX1/TXD3/SDA1
P0.1/CAN_TX1/RXD3/SCL1
P0.2/TXD0
P0.3/RXD0
P0.4/I2SRX_CLK/CAN_RX2/CAP2.0
P0.5/I2SRX_WS/CAN_TX2/CAP2.1
P0.6/I2SRX_SDA/SSEL1/MAT2.0
P0.7/I2STX_CLK/SCK1/MAT2.1
P0.8/I2STX_WS/MISO1/MAT2.2
P0.9/I2STX_SDA/MOSI1/MAT2.3
P0.10/TXD2/SDA2/MAT3.0
P0.11/RXD2/SCL2/MAT3.1
P0.12/PORT_PWR-BN/MISO1/AD0.6
P0.13/LED_BN/MOSI1/AD0.7
P0.14/HOST_EN-BN/DEVICE_EN-BN/SSEL1
P0.15/TXD1/SCK0/SCK
P0.16/RXD1/SSEL0/SSEL
P0.17/CTS1/MISO0/MISO
P0.18/DCD1/MOSI0/MOSI1
P0.19/DSR1/MCICLK/SDA1
P0.20/DTR1/MCICMD/SCL1
P0.21/RI1/MCIPWR/CAN_RX1
P0.22/RTS1/MCIDAT0/CAN_TX1
P0.23/AD0.0/I2SRX_CLK/CAP3.0
P0.24/AD0.1/I2SRX_WS/CAP3.1
P0.25/AD0.2/I2SRX_SDA/TXD3
P0.26/AD0.3/AOUT/RXD3
P0.27/SDA0
P0.28/SCL0
P0.29/USBD+A
P0.30/USBD-A
P0.31/USBD+B
USBD-B
P4.0/A0
P4.1/A1
P4.2/A2
P4.3/A3
P4.4/A4
P4.5/A5
P4.6/A6
P4.7/A7
P4.8/A8
P4.9/A9
P4.10/A10
P4.11/A11
P4.12/A12
P4.13/A13
P4.14/A14
P4.15/A15
P4.24/OEN
P4.25/WEN
P4.28/BLS2/MAT2.0/TXD3
P4.29/BLS3/MAT2.1/RXD3
P4.30/CSN0
P4.31/CSN1
P0.[0..7]
P0.[8..15]
P0.[16..23]
P0.[24..31]
USBD-B
P4.[0..7]
P4.[8..15]
P4.[24..31]
KEIL SOFTWARE
Project-Version: MCB2300-v3.1
Date / Time: 11.07.2007 17:42:13
Sheet: 1/6


P3
COM1
P4
COM0
IC2
MAX563
V+
V-
EN
SHDN
C1+
C1-
C2+
C2-
T1OUT
T2OUT
R1IN
R2IN
T1IN
T2IN
R1OUT
R2OUT
GND
3,3V
C8
C9
C6
C7
100n
+3,3V
P0.15
P0.2
P0.16
P0.3
RST
PMLL4448
D3
47k
R18
33k
R17
22k
R20
S1
RESET
C20
100n
RST
J9
BC850
T1
D2
PMLL4448
BC850
T2
33k
R19
PMLL4448
D4
S2
INT0
C21
100n
22k
R21
J8
ISP
J10
INT0
P2.10
IC3
LM386M
VIN
VCC
+IN
-IN
VOut
G1
G8
GND
BP
P0.26
J3
AOUT
100n
C10
R11
10k
1k
R10
10R
R9
C12
RM220/35
Loudspeaker
C11
100n
C13
33n
K2
IN
K
OUT
GND
IC8
LD1117S50
C44
100n
C24
10uF/16V
D7
BAT60A
+5V
K1
IC1
LD1117S33
C3
10uF/16V
+3,3V
POWER
R8
1K
D1
SM4004
C19
100n
P0.30
J1
D-
C2
18p
R1
33R
P0.29
J2
D+
C1
18p
R2
33R
R3
1,5k
D8
BAT60A
D5
BAT60A
C4
10uF/16V
P2
VIN
J7
VBUS
4.7k
R71
R73
220k
P1.30
R56
1M
C43
4,7n
P1
USB-PLUG
D-
D+
+3,3V
10k
R7
Q1
PMBTA92
P2.9
J5
UMODE
R6
2,2k
R5
680R
R42
680R
USC
UGL
P1.18
+3,3V
R30
22k
IC9
74LVC244
1OE
2OE
1A1
1A2
1A3
1A4
2A1
2A2
2A3
2A4
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
+3,3V
GND
P2.7
P2.6
P2.5
P2.4
P2.3
P2.2
P2.1
P2.0
R22
R23
R24
R25
R26
R27
R28
R29
680R
J11
LED
KEIL SOFTWARE
Project-Version:
MCB2300-v3.1
Date / Time:
11.07.2007 17:42:13
Sheet:
2/6

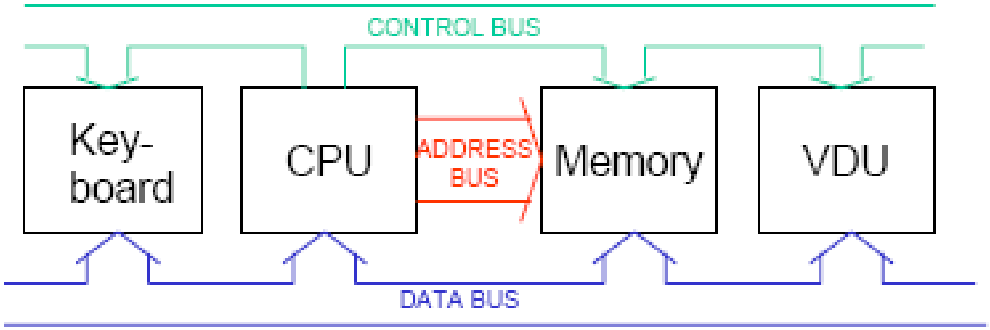
ARM7TDMI microprocessor

- 74,209 transistors
- 32 bit address bus
- 0.13 µm gate length
- upto 133 MHz clock
- 0.26 mm^2 die area
- max. power 8 mW

T: 16-bit **T**humb code
D: on-chip **D**ebug support
M: an enhanced **M**ultiplier
I: Embedded**I**CE hardward to give
on-chip breakpoint and
watchpoint support

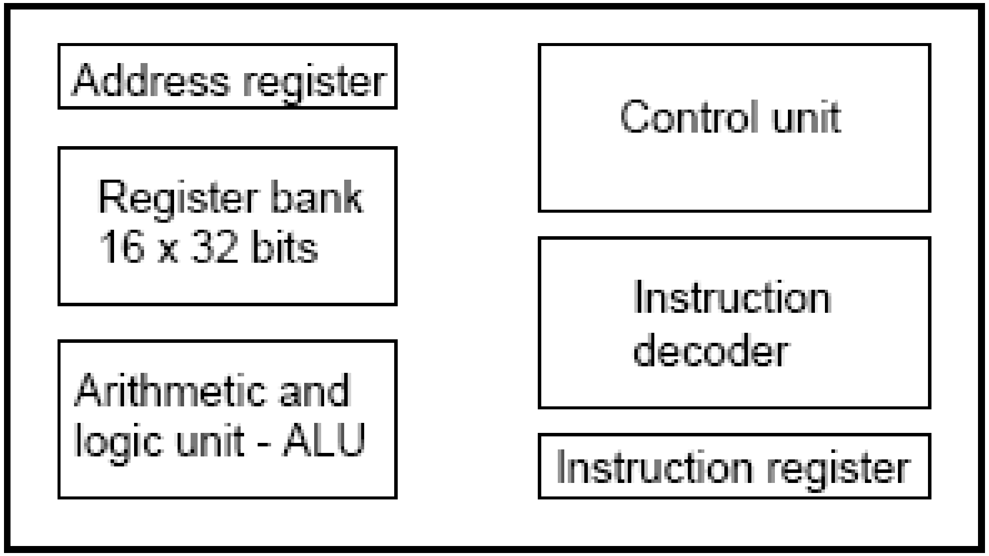
A Third Bus

- In addition to the data bus and control bus there is a third bus called the address bus.
- The address bus is used by the CPU to determine which location in memory is sending or receiving data.



The ARM7 core

- The basic building blocks of the ARM7 core are:

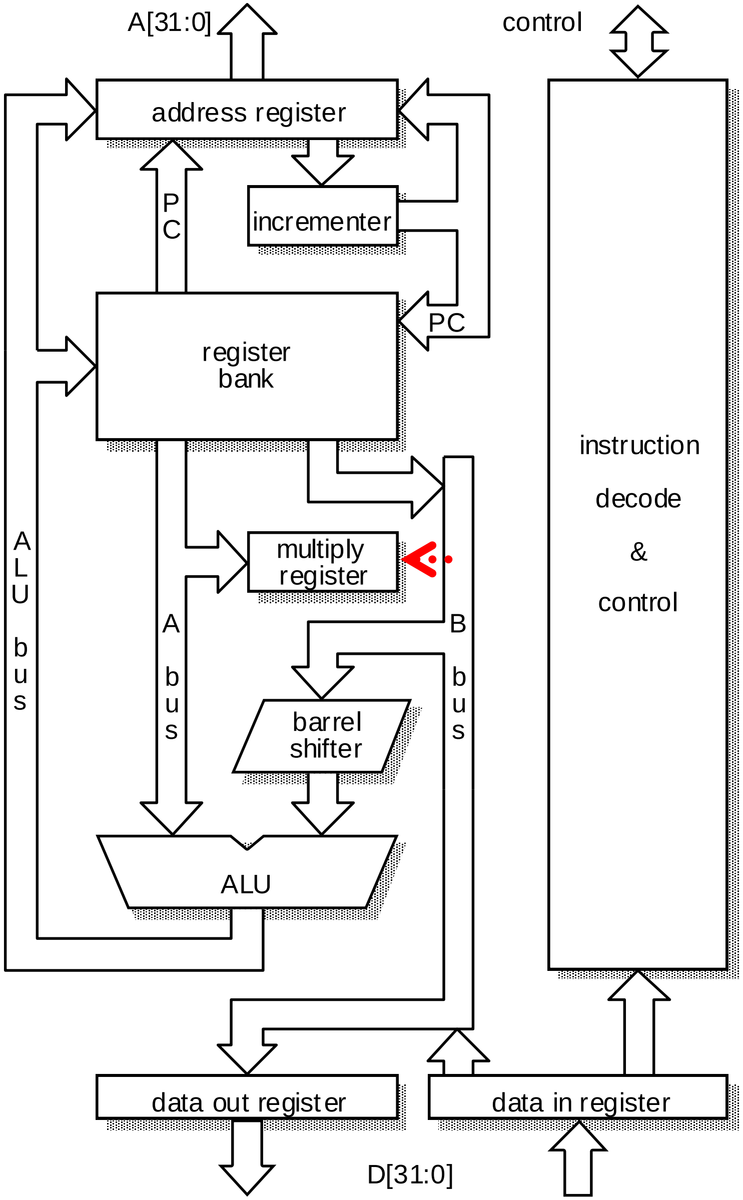


ARM7 CPU:

many internal registers;
  address register
  and register bank.

3 internal datapaths;
  A bus,
  B bus,
  ALU bus.

additional ALU functionality;
  multiplier
  and barrel shifter.



The barrel shifter

- The barrel shifter is a very useful feature of the ARM7 microprocessor which allows bit patterns to be rotated.
- E.g. the instruction

  MOV r1, r2, LSL #5

  - would take the bit pattern in register r2 and shift it 5 places to the left before placing it in register r1. So if r2 held:

    1010 0101 1100 0011 1001 0110 1110 0111

  - after the instruction was executed r1 would hold:

    1011 1000 0111 0010 1101 1100 1110 0000

ALU adder design

- ripple-carry adder
- carry look-ahead
- carry select adder

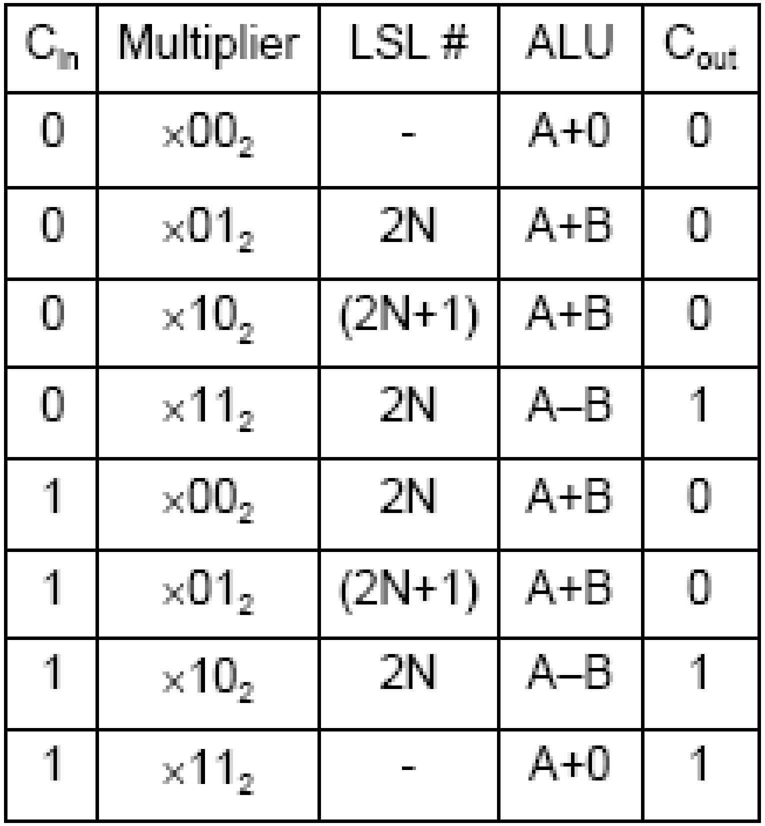
Booth’s algorithm: Nth cycle

Booth’s algorithm works by replacing ×3 by ×(4-1) so that each cycle multiples by a 2 bit value.

N bit multiplications can be completed in N/2 cycles.

The table right summarizes the action on the Nth cycle.

Table 4.3 in Furber also replaces ×2 by ×(4-2) for $C_{in=0}$ as this makes the control logic slightly simpler.



C_{in}	Multiplier	LSL #	ALU	C_{out}
0	$\times 00_2$	-	A+0	0
0	$\times 01_2$	2N	A+B	0
0	$\times 10_2$	(2N+1)	A+B	0
0	$\times 11_2$	2N	A–B	1
1	$\times 00_2$	2N	A+B	0
1	$\times 01_2$	(2N+1)	A+B	0
1	$\times 10_2$	2N	A–B	1
1	$\times 11_2$	-	A+0	1

- ✓ Organization/Architecture
- ✓ **Mnemonics**
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Program counter.

- Instructions are stored in memory so:
  - How does the CPU know the memory address for the next instruction in the computer program?
  - Register r15 always holds the memory address of the next instruction to be executed.
  - An alternative name for register r15 is the ‘program counter’.

Instructions

- For a human to perform a task he/she needs instructions e.g.
  - – to knit a jumper
    - the instructions are the knitting pattern
  - – to bake a cake
    - the instructions are in the recipe.
- Computers also need instructions
  - – so that the data is processed into information correctly.

Instructions stored in memory.

- Instructions are 32 bits long, whereas memory locations are 8 bits long so one instruction occupies four locations in memory, e.g.
  - one instruction would be stored in 4 memory locations with addresses 0x00008000,0x00008001, 0x00008002 and 0x00008003.
  - The next instruction would be stored at addresses 0x00008004, 0x00008005, 0x00008006 and 0x00008007 and so on.

Simple instructions.

- One of the simplest instructions is to move a value into a register e.g.
  - move 114 into register r12
  - The machine code for this instruction is 0xE3A0C072
  - After the instruction is executed
    - register r12 will hold the value 0x00000072 (hexadecimal for 114) and
    - value in register r15, the program counter, will have increased by 4.
    - All other registers remain unchanged.

Machine code.

- Look at the machine code for the instruction;
  - move 114 into register r12, again.
    - 0xE3A0C072
    - 1110 0011 1010 0000 1100 0000 0111 0010
  - The value 114 is given in the least significant byte
    - 0xE3A0C072
    - 114 in decimal is 72 in hexadecimal.
  - Register r12 is given by the 5th digit
    - 0xE3A0C072
    - 12 in decimal is C in hexadecimal.

Mnemonics

- In general nobody remembers all of the machine code for any particular processor (or indeed any).
- Instead we use mnemonics
  - mnemonics are words or phrases which are easy to remember and
  - can replace something which is difficult to remember.
- The instruction to move the value 114 into register r12 has the mnemonic
  - MOV r12, #114.
  - This is much easier to remember than 0xE3A0C072

Instructions for Arithmetic

- The ARM7 can add, subtract and multiply numbers (but not divide).
- The mnemonic for add the value in register x to the value in register y and place the sum in register z is: ADD rz, ry, rx
- E.g. to add the value in register r1 to the value in register r2 and leave the sum in register r3
  - the mnemonic is: ADD r3, r2, r1
  - We don't need to know the machine code.

Instructions using logic

- As well as arithmetic the ARM7 can also do logic such as AND and OR.
- AND value in rx with value in ry and leave the result in rz has the mnemonic:

  AND rz, ry, rx

- OR value in rx with value in ry and leave the result in rz has the mnemonic:

  ORR rz, ry, rx

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ **Flags**
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Flags

Flags are used to give a signal that something either has or has not happened.

For example, the flag is raised to half mast on in front of buildings as a signal that someone important is dead.

Microprocessors also use “flags” to signify what has happened.

Flags and Processors

Flags are essentially 1 bit memory devices and different microprocessors have different flags.

The ARM microprocessor has four flags that are commonly found in the majority of microprocessors.

These flags are the zero flag (Z), the negative flag (N), the carry flag (C), and the overflow flag (V).

Flags are also know as ‘condition codes’.

What are flags used for?

Flags are used in two ways:

The main use of flags is to determine if another instruction is executed or not. This is called conditional execution.

The carry flag can also be used in some arithmetic instructions as an additional value (we will return to this later).

Branches

- An important use of conditional execution is in branches.
- Conditional branches are used when a processor performs one function rather than another
  - e.g.
    - display text messages rather than receive an incoming call.
- First consider the unconditional branch.

Flags - summary.

- The zero flag, Z, is set when the result (not including any carry out) is 0x00000000.
- The negative flag, N, is set when the most significant bit of the 32 bit result is 1.
- The carry flag, C, is set when the result (taken as an unsigned integer) is greater than $(2^{32} - 1)$.
- The overflow flag, V, is set when the result (taken as a two’s compliment number) is greater than $(2^{31} - 1)$ or less than -2^{31}

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ **Representations**
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Representing characters

- As well as manipulating numbers, computers also manipulate characters e.g.
  - in a word processing package like Microsoft Word.
  - Characters must be coded in binary before computers can process them
  - the standard coding for characters is ASCII (American Standard Code for Information Interchange).

Negative numbers

- There are two main methods for representing negative numbers in microprocessors.
- These are:
  1) Sign magnitude.
  2) Two’s complement.
- In each case the most significant bit - ‘m.s.b.’ - indicates the sign (1 for -ve and 0 for +ve).

2’s complement

- The two’s complement method automatically sets the m.s.b. or ‘sign bit’ to 1.
- The following method can be used to find a 2’s complement representation of a negative number,
  - e.g.
    - -20640
    - First find the positive value: 0x000050A0 or 0000 0000 0000 0000 0101 0000 1010 0000_2
    - Next invert all bits (0→ 1, 1 → 0).
      - 1111 1111 1111 1111 1010 1111 0101 1111_2 or
      - 0xFFFFAF5F.
    - And then add 1 ➔0xFFFFAF60

Floating point numbers

- The floating point format uses the equation A × 2^n to represent numbers.
- Using the IEEE 754 standard the A value has a sign and 24 bits and the exponent value, n, has 8 bits.
- The number $5,637,144,576_{10}$ (= $2^{32} + 2^{30} + 2^{28}$) is too big for a 32 bit unsigned integer.
- In binary this number is
  - $1\ 0101\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$
- First we must normalize this number as follows:

5,637,144,57610 in IEEE 754

- 1 0101 0000 0000 0000 0000 0000 0000 0000_2= 1.0101 0000 0000 0000 0000 000_2 × 2^{32}
- The exponent is 32_{10} = 100000_2 but in the IEEE 754 format we must add 127_{10} = $1111111_2$to this to give 10011111_2 (=159_{10})
- In 32 bits this becomes:

  0100 1111 1010 1000 0000 0000 0000 0000
- Note that the normalization means that the m.s.b. of the A value is always 1 so this can be omitted.
- The m.s.b. of the 32 bit word is a sign bit.

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ **Subroutines**
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Subroutines

- Many microprocessor programs include groups of instructions which are repeated many times.
- It is wasteful of memory to include these instructions in the program again and again.
- Instead they can be included once in a special structure known as a subroutine.
- The main program branches to the start of the subroutine when it requires those particular instructions.
- At the end of the subroutine there is another branch back to the main program.

Problem

- At the end of the subroutine how does the processor know which instruction to return to?
- This problem is solved by using a ‘link register’.
- A link register holds the memory address of the instruction in the main program to which the subroutine returns to.
- Register r14 in the ARM microprocessor is designated as the link register and ‘r14’ can be replaced by ‘lr’ in mnemonics.

Branch and link

- The mnemonic for ‘branch and link’ is BL;
  - a simple branch with mnemonic B does not update the link register.
- To return from the subroutine the value in the link register is moved into the program counter;
  - MOV pc, lr.
- What happens if a branch and link occurs in a subroutine?
  - The value held in the link register will be overwritten by a new return address so before one subroutine calls another the link register value must be stored elsewhere.

Nested subroutines

- Complicated programs will have several subroutines and some subroutines will call other subroutines
  - this is standard practice and it is known as nesting.
- If subroutine A calls subroutine B the link register can not store the return address for both subroutines at the same time.
- In order to preserve the return address of all subroutines an area of computer memory called the stack is used.
- The stack is a ‘last in first out’ queue.

The stack

- The stack is a last in first out queue
  - that means whatever data was added to the stack (‘pushed’) last is taken from the stack (‘popped’) first.
  - E.g. if the values pushed onto a stack were 0x00FF, 0xFF00, 0xAAAA in that order then they would be popped from the stack in the reverse order.
- In memory the stack is held as a list:

top of stack	0xAAAA
	0xFF00
bottom of stack	0x00FF

Another problem

- The stack is a dynamic memory structure
  - Meaning that the amount of data held in the stack can vary.
- The bottom of the stack can be fixed at a particular memory address.
- How do we know where the top of the stack is?
  - We need to know so that we can pop the correct data.
- Another register is used to identify the top of the stack - it is known as the stack pointer.

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ **Interrupt**
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Interrupts

- Many applications for microprocessor systems require things to happen when an event occurs unexpectedly.
- These unexpected events can interrupt the normal operation of the microprocessor.
- During an ‘interrupt’ the execution of the main program is halted and a special program is executed instead.
- When the interrupt program has finished, the microprocessor returns to the main program.

Interrupts

- Interrupts provide a very convenient method for dealing with events and this method is often used for events that are not unexpected
  - e.g.
    - when a mobile phone receives an incoming call.
- An interrupt is triggered when the microprocessor receives a (voltage) signal on a special connection within the control bus.
- The ARM7 has two types of interrupts,
  - a normal interrupt (IRQ) and
  - a fast interrupt (FIQ).

Interrupt Handling

- When an interrupt occurs the following happens:
  ① The registers for IRQ mode or FIQ mode are activated.
  ② The current program status register, CPSR, (this contains the flags and other information) is saved into a saved program status register, SPSR.
     a) There are two SPSR registers, one for each mode.
  ③ The return address of the next instruction to be executed in the main program is stored in the link register for the appropriate mode.
  ④ The program counter is set to either 0x00000018 for an IRQ or 0x0000001C for a FIQ.

Interrupt Vectors

- The first instruction executed by an interrupt program is the instruction stored at memory address 0x00000018 for an IRQ or 0x0000001C for a FIQ.
- These memory addresses are known as ‘vectors’.
- For an IRQ the instruction at address 0x00000018 must be a branch to another part of memory because the following memory location, 0x0000001C contains the first instruction of the FIQ handler.

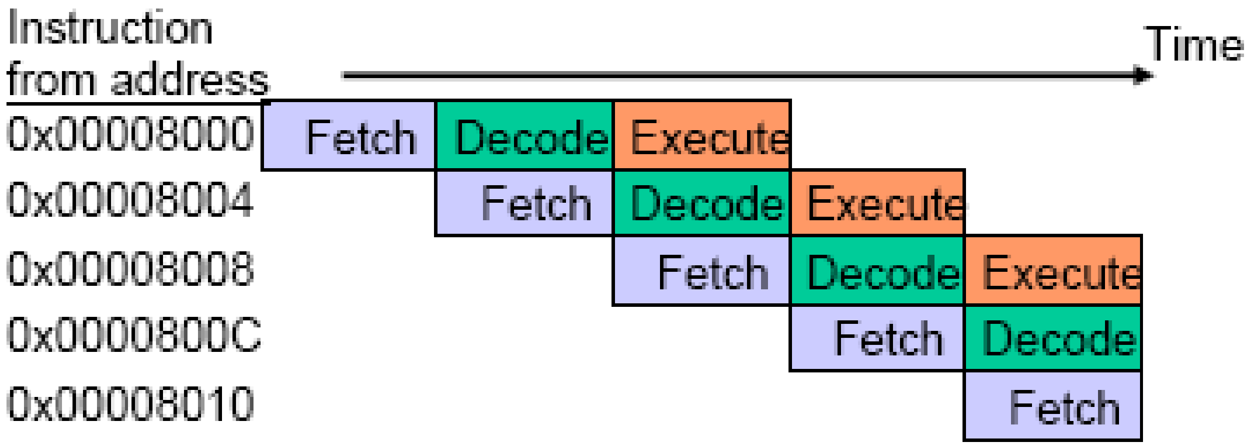
- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ **Pipeline**
- ✓ Encoding
- ✓ Memory
- ✓ Processor Modes

Instruction Pipelines

- Instruction pipelines are an important feature of all modern microprocessors.
- For the same basic speed of transistor operation, an n stage instruction pipeline allows the microprocessor to execute up to n times as many instructions in a given time.
- The ARM7 microprocessor has a three stage pipeline
  - one stage for each of the CPU cycles;
    - fetch, decode and execute.

Instruction Pipelines

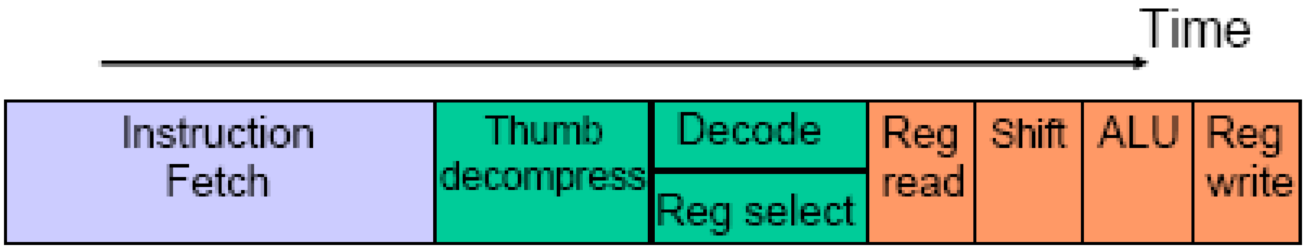
- In a three stage pipeline, the CPU can simultaneously execute an instruction, decode the next instruction and fetch the next instruction.



Instruction from address	Time				
0x00008000	Fetch	Decode	Execute		
0x00008004		Fetch	Decode	Execute	
0x00008008			Fetch	Decode	Execute
0x0000800C				Fetch	Decode
0x00008010					Fetch

ARM7 3 stage pipeline: detail

In each stage of the ARM7 pipeline, several things happen; normally consecutively:



Time

Instruction Fetch

Thumb decompress

Decode

Reg select

Reg read

Shift

ALU

Reg write

FETCH

Instruction fetched from memory.

DECODE

Thumb to ARM decompression.
Decode.
Register selection

EXECUTE

Register read.
Barrel shift.
ALU operation.
Register write.

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ **Encoding**
- ✓ Memory
- ✓ Processor Modes

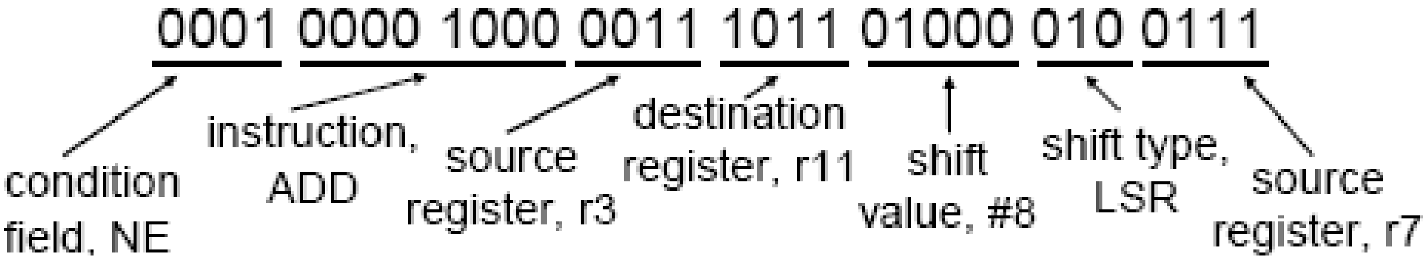
ARM7 pipeline example

Consider a typical three operand ARM7 instruction e.g.

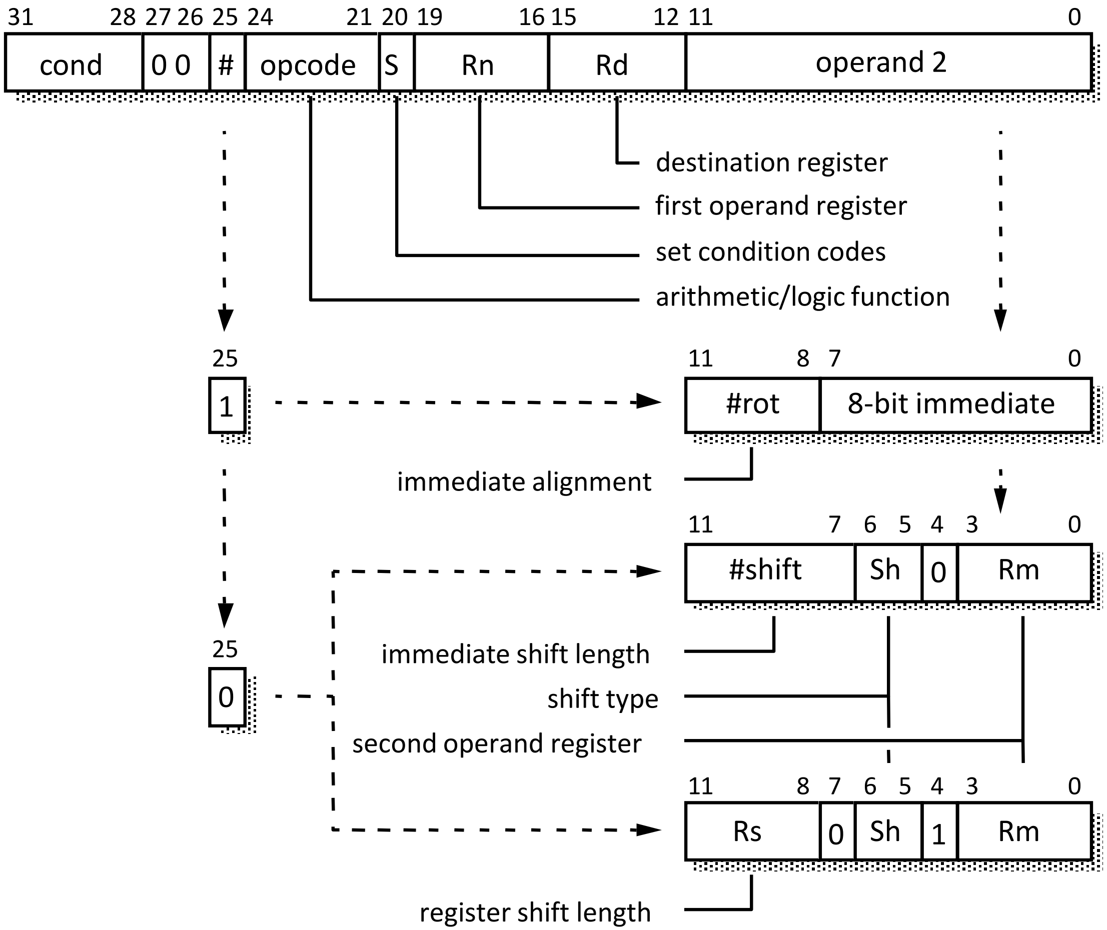
ADDNE r11, r3, r7 LSR #8

Add the contents of register r3 to the contents of register r7 shifted right by 8 bits and put the sum in register r11 if the zero flag is clear (Z=0).

The machine code for this instruction is 0x1083B427



Data processing instruction binary encoding



For Internal Use Only!

ARM data processing instructions

Opcode [24:21]	Mnemonic	Meaning	Effect
0000	AND	Logical bit-wise AND	Rd := Rn AND Op2
0001	EOR	Logical bit-wise exclusive OR	Rd := Rn EOR Op2
0010	SUB	Subtract	Rd := Rn - Op2
0011	RSB	Reverse subtract	Rd := Op2 - Rn
0100	ADD	Add	Rd := Rn + Op2
0101	ADC	Add with carry	Rd := Rn + Op2 + C
0110	SBC	Subtract with carry	Rd := Rn - Op2 + C - 1
0111	RSC	Reverse subtract with carry	Rd := Op2 - Rn + C - 1
1000	TST	Test	Scc on Rn AND Op2
1001	TEQ	Test equivalence	Scc on Rn EOR Op2
1010	CMP	Compare	Scc on Rn - Op2
1011	CMN	Compare negated	Scc on Rn + Op2
1100	ORR	Logical bit-wise OR	Rd := Rn OR Op2
1101	MOV	Move	Rd := Op2
1110	BIC	Bit clear	Rd := Rn AND NOT Op2
1111	MVN	Move negated	Rd := NOT Op2

- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ **Memory**
- ✓ Processor Modes

What is computer memory?

- Computer memory is a very big sequential logic circuit made up of thousands or millions of simple logic gates, such as a D type latch, which can remember a 0 or a 1, that is one bit of data.
- Groups of these gates are collected together in a memory ‘location’.
- There are typically 8 bits of data in one location.
- Each memory location has a unique memory address.

Memory organization.

- Taking the ARM7TDMI microprocessor as an example
  - at each memory location
  - it has 8 bits of data
    - 8 bits is known as a byte.
- The ARM is a 32 bit processor and addresses are 32 bits long from 0x00000000 to 0xFFFFFFFF.
  - That means there can be up to 4,294,967,296 (or 2^{32}) different memory locations all with a unique memory address.
- In practice not all addresses are used for memory.

Addressing modes

- The concept of addressing mode is common to all microprocessors and an understanding of addressing mode is important for users of microprocessors.
- The ARM7 microprocessor supports many different addressing modes and we will concentrate on just four.

Load and Store

- When a load instruction is executed
  - the data travels from memory to register
- whereas when a store instruction is executed
  - the data travels from register to memory.
- Each memory location holds one byte or 8 bits of data whereas each register holds 4 bytes of data.
- So LDR and STR use four consecutive memory addresses but which byte goes to which location?

Little endian

- Microprocessors can be either ‘little endian’ or ‘big endian’
- If the processor is ‘little endian’ then the instruction:
  - STR r6, [r11]
  - with 0xFFAABB11 in r6 and 0x00008000 in r11
  - would store
    - byte 0x11 at address 0x00008000
    - 0xBB at 0x00008001
    - 0xAA at 0x00008002
    - 0xFF at 0x00008003

Big endian

- Whereas if the processor is ‘big endian’ then the instruction:
  - STR r6, [r11]
  - with 0xFFAABB11 in r6 and 0x00008000 in r11
  - would store
    - byte 0xFF at address 0x00008000
    - 0xAA at 0x00008001
    - 0xBB at 0x00008002
    - 0x11 at 0x00008003

Types of Silicon Memory

- Silicon IC memory can be classified as either read-only or read-write.
- Read-only memory (or ROM) is used for the operating system on a desktop computer or an application program in an embedded system such as a mobile phone.
- Read-write memory (which is normally called RAM) is used for an application program on a desktop computer and for temporary data.

Types of ROM

- There are several generations of silicon ROM.
- Programmable ROM (PROM) could be electrically programmed once only and then the contents were fixed.
- Erasable PROM (EPROM) could also have its contents erased by exposure to ultraviolet (UV) light and could then be reprogrammed.
- Electrically erasable PROM (EEPROM) could have its contents erased by an electrical signal.

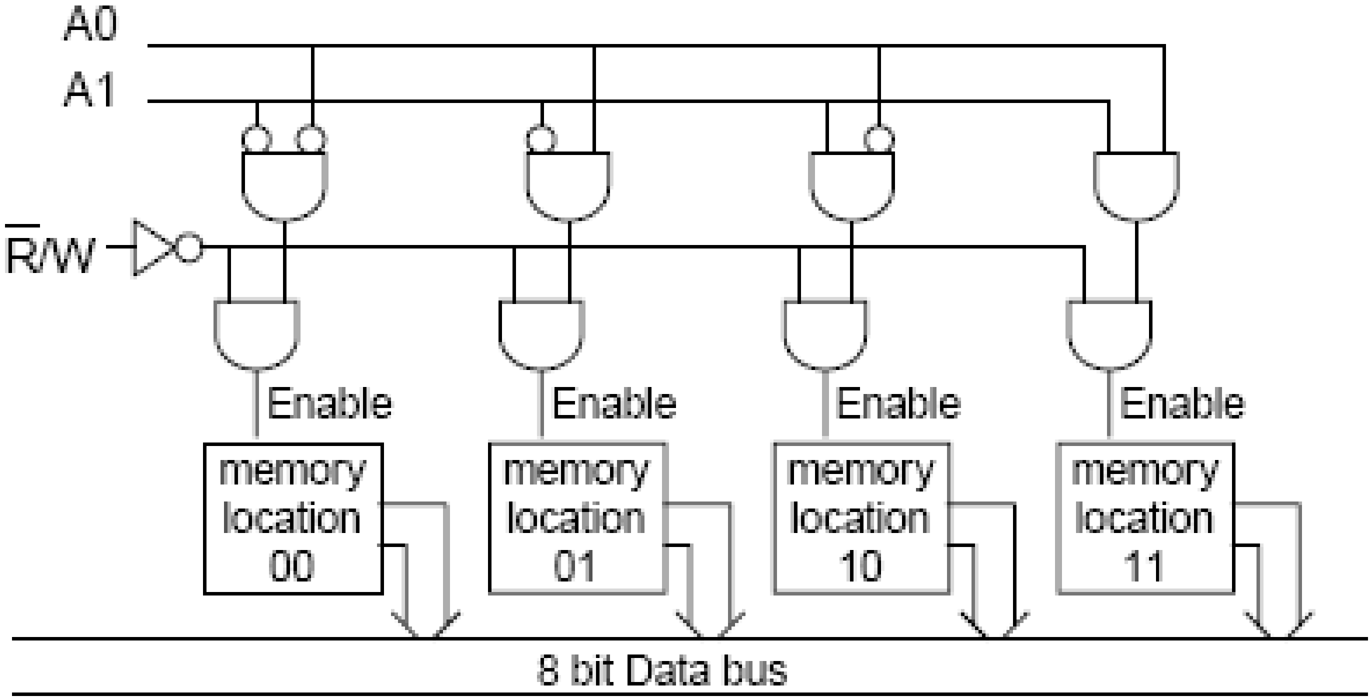
Types of ROM

- Flash Erasable PROM (FEPROM or flash memory) is similar to EEPROM but erasing could only be done over a significant part of (maybe all) the integrated circuit.
- All ROM is non-volatile - meaning the information stored in ROM memory is not lost when it is disconnected from a power source. The data stored in ROM will remain almost indefinitely e.g. many tens of years.
- In contrast the contents of RAM memory will be lost almost as soon as power is switched off - RAM is volatile.

Cache memory

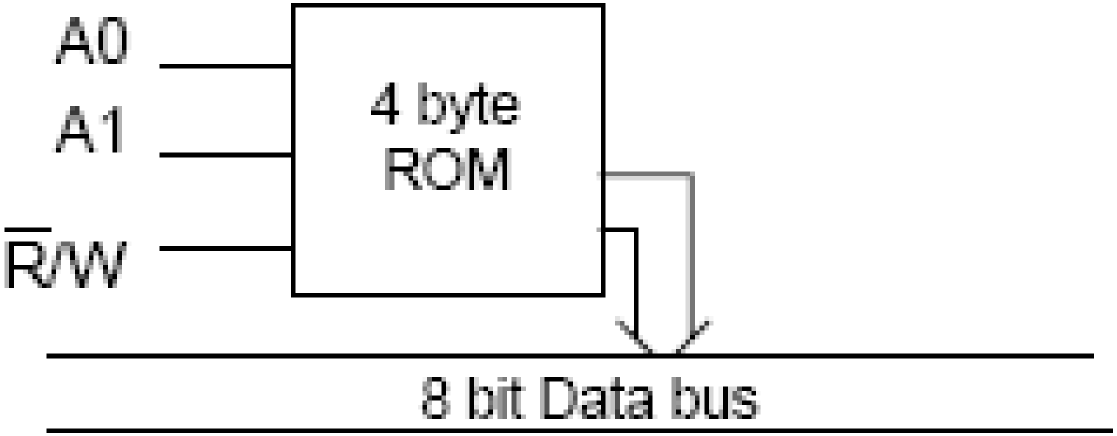
- When data is read from, or written to, main memory a copy of the data is saved in the memory cache (along with the main memory address).
- When a subsequent read occurs the cache can be checked to see if the required data is already there.
- If it is (a cache ‘hit’) then it can be supplied immediately.
- If not (a cache ‘miss’) then it must be fetched from main memory.

Simple 4 byte ROM addressing



Simple ROM addressing

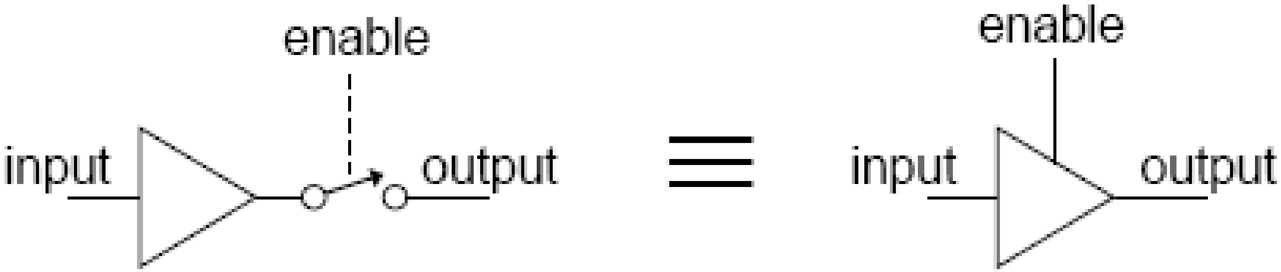
- So memory location 00 is ‘enabled’ and connected to the data bus only when R*/W = 0, A0 = 0 and A1 = 0.
- The previous circuit can be represented by a single block as follows:



Tri-state gates

- Only one memory location should be connected to the bus at one time – one device can not drive a logic 0 onto a bus line at the same time as another device drives a logic 1 onto the same line.
- This is achieved using ‘tri-state gates’ which have an ‘enable’ input – when not enabled the output acts like an ‘open circuit’ which is referred to as ‘high impedance’.

Tri-state gates

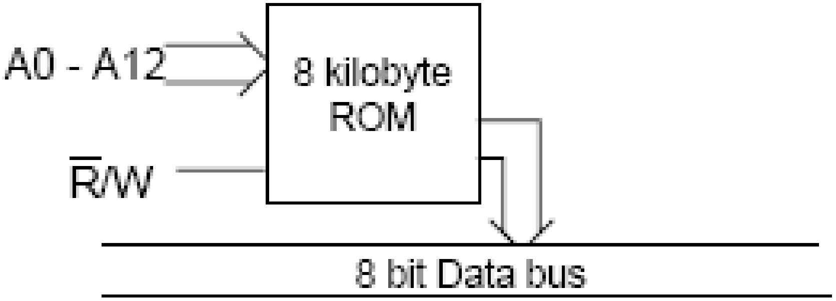


enable	input	output
0	0	high-impedance
0	1	high-impedance
1	0	0
1	1	1

Simple ROM addressing

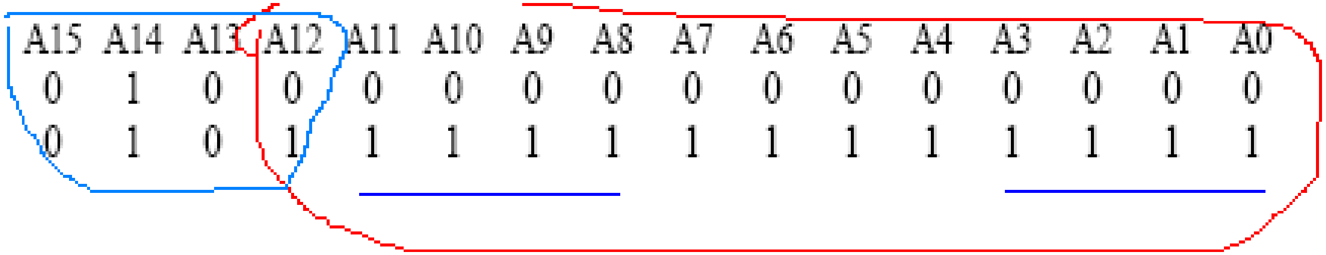
- Bigger memories would have similar connections but there would be more address lines.
- A 2^N byte memory would need N address lines:
- E.g. a 4 kilobyte memory would need 12 address lines.

8 kilobyte	13
16	14
etc.	



Which address?

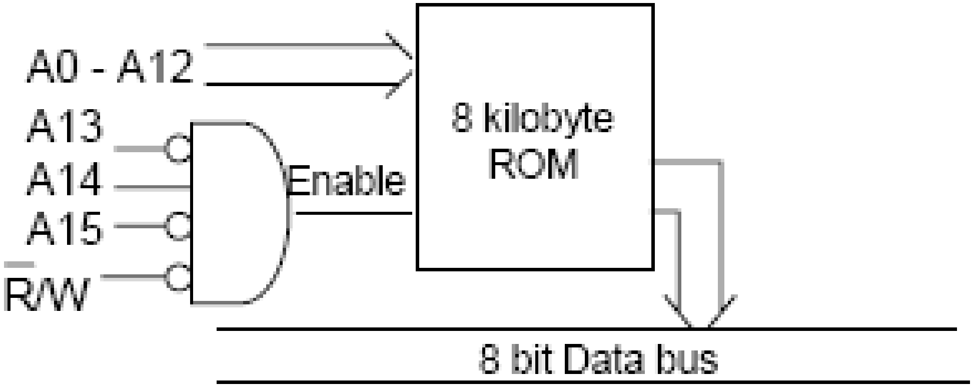
- If an 8 kilobyte memory chip is connected to a processor with a 16 bit address bus whereabouts is the memory located within in the 64 kilobyte address space?
- The lowest N address lines are connected to the 2^N byte memory; in this case A0 to A12. The other address lines, A13 to A15, must be decoded e.g. if the memory range is from 0x4000 to 0x5FFF or in binary:



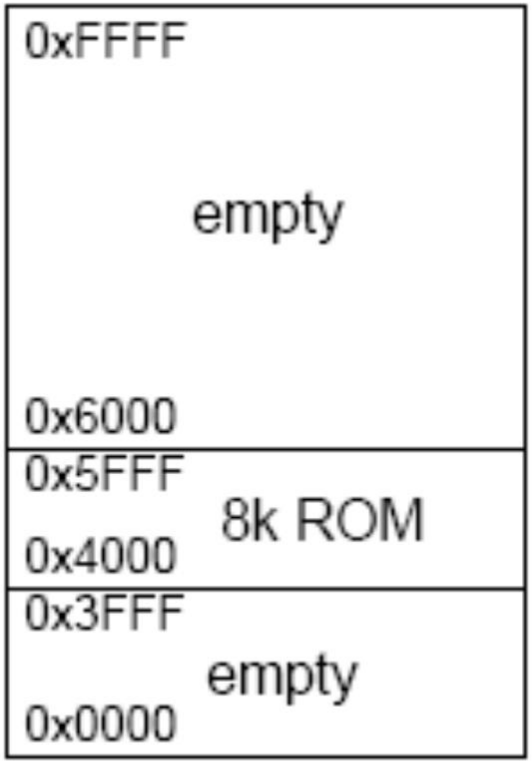
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

8 kilobyte ROM addressing

- The memory is only enabled when A13 = 0, A14 = 1, A15 = 0 and R*/W = 0.

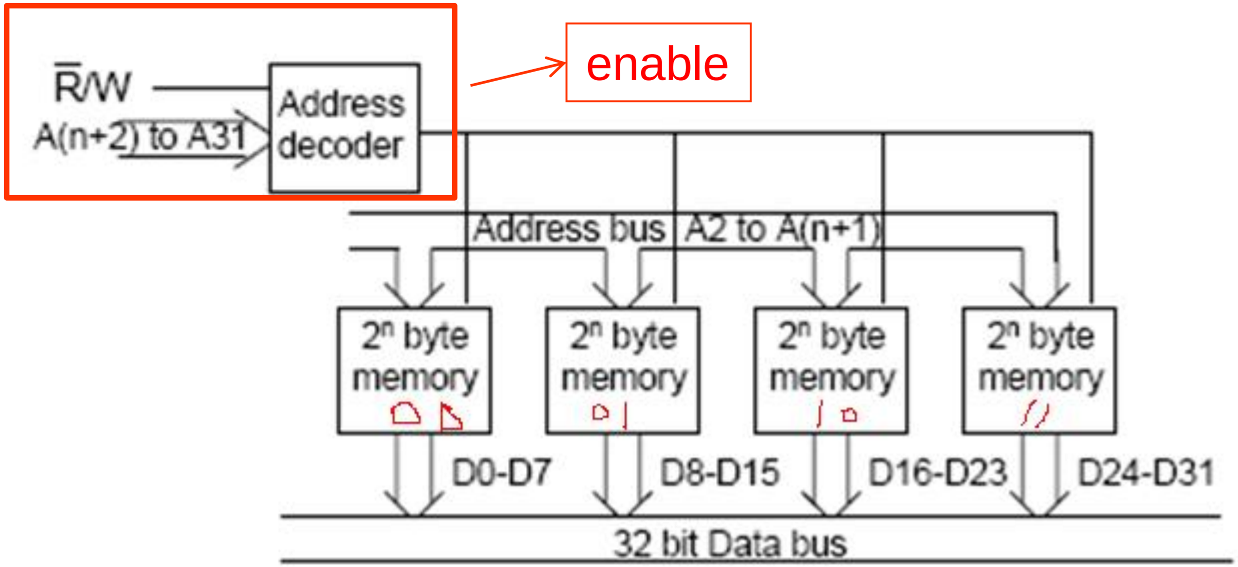


- If the address bus is 32 bits wide then all the address lines not connected to the memory chip, A13 to A31, are decoded.



Connecting to a 32 bit data bus

- Memory with 8 data bits output can be connected to a 32 bit data bus by using 4 separate memory chips; one for each byte



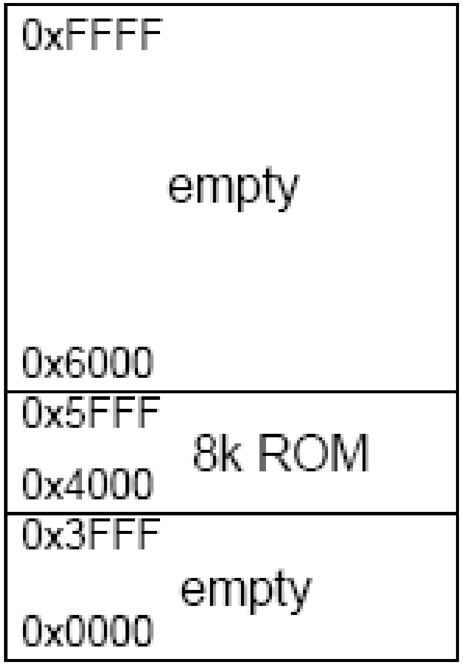
$A_0 – A_{n+1}$: memory size

$A_{n+2} – A_{31}$: location in the memory map

A0 :
A1 :

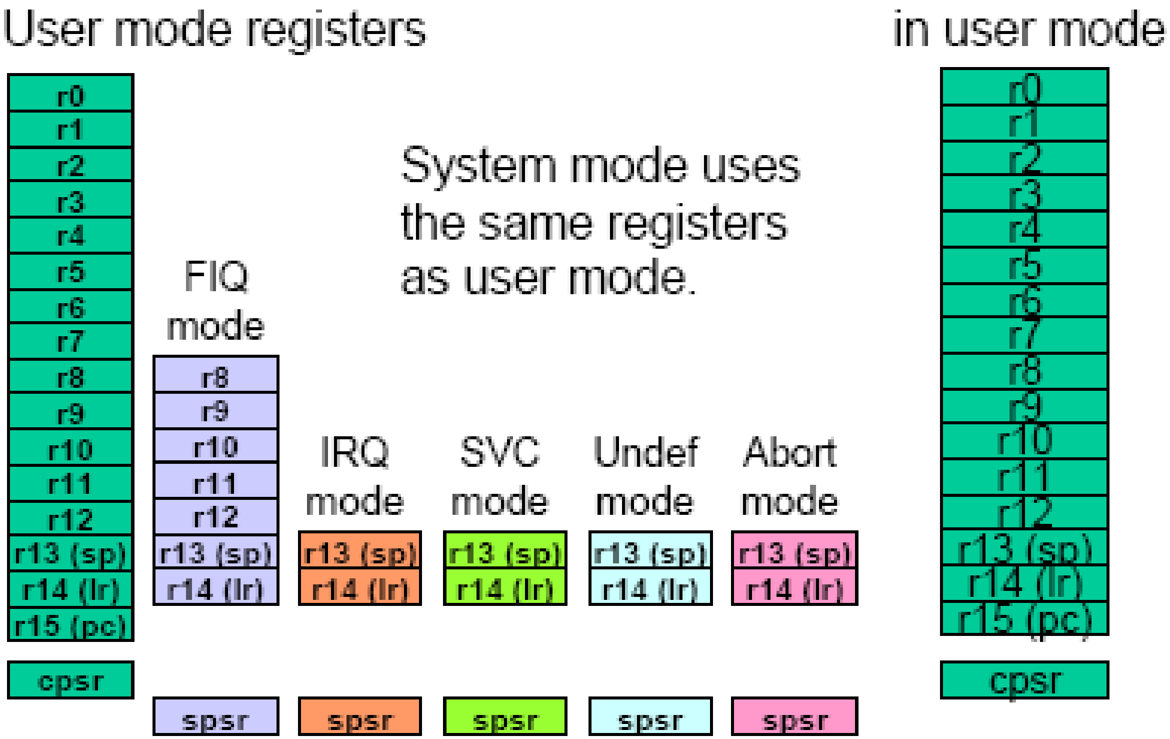
Memory maps

- The location of a memory chip within the complete addressable space can be represented by a memory map.
- E.g. for the 8 kilobyte ROM memory between addresses 0x4000 and 0x5FFF in a 64 kilobyte addressable space:



- ✓ Organization/Architecture
- ✓ Mnemonics
- ✓ Flags
- ✓ Representations
- ✓ Subroutines
- ✓ Interrupt
- ✓ Pipeline
- ✓ Encoding
- ✓ Memory
- ✓ **Processor Modes**

Different registers in each mode



User mode registers

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)

cpsr

FIQ mode

r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)

spsr

IRQ mode

r13 (sp)
r14 (lr)

spsr

SVC mode

r13 (sp)
r14 (lr)

spsr

Undef mode

r13 (sp)
r14 (lr)

spsr

Abort mode

r13 (sp)
r14 (lr)

spsr

System mode uses the same registers as user mode.

in user mode

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)

cpsr