Unit 9:
The Motorola 68000 µP
1. Overview of the 68000 µP
2. Programming model, assembly, addressing modes, stack
3. 68000 Hardware interfacing, bus arbitration
4. Read/Write cycles
5. Memory organization
6. Memory interfacing
7. I/O interfacing
8. Exception processing, hardware interrupts

Reading: Antonakos, chapters 1,2,3,4,7,8,9,12
Motorola 68000 µP

- 6800 µP - introduced in 1974, 8-bit.
  - NMOS (N-channel MOS technology)
  - 68K transistors?
  - 64 Pin DIP (8086 is 40) → No multiplexing.
  - Internal architecture is 32 bits (ALU is 16 bits wide).
  - 23 bit (physical) address bus $A_1 - A_{23}$. No $A_0$.
    - (24 bit effective address bus with LDS/UDS...)
  - 16 bit data bus.
- Operands:
  - Bytes
  - Words (16-bits)
  - Long words (32-bits)
Motorola 68000 µP

• **68008** - 8 bit data bus, 20 bit address bus.
• **68010** - 1982. Added virtual memory support.
• **68020** - 1984. Fully 32 bit. 3 stage pipeline.
  – 256 byte cache. More addressing modes!
• **68030** - 1987. Integrated MMU into chip.
• **68040** - 1991. Harvard architecture with two 4KB caches. FP on chip. 6 stage pipeline.
• **68060** - 1994. Superscalar version. 10-stage pipeline. 2 integer, 1 fp unit. 8k caches. 4-5W.
• **Coldfire** - 1995. Embedded version. Stripped out funky addressing modes.
Motorola 68000 µP

• Used in:
  – Apple Macintosh (then PPC, now Intel!!).
  – Atari 520ST and 1040ST (Defunct).
  – Amiga (Defunct).
  – Early Sun workstations (Now SPARC).
  – NeXT (Defunct, purchased by Apple 1996, MAC OS X came from ‘NeXTStep’).

• 68000 architecture has user/supervisor modes.
  – protects operating system.
  – supports multitasking and multiprocessing.
Motorola 68000 µP – Programming Model

Data Registers

Address Registers

Status Register

User Stack Pointer

Supervisor Stack Pointer

Program Counter
Motorola 68000 µP - Assembler

- **format:** INST SRC, DST
- **Prexes:**
  - `%` binary
  - `$` Hexadecimal
  - `#` Immediate.
- **Attach size to instruction, e.g.**:
  - `move.b` byte
  - `move.w` word
  - `move.l` long word
- `( )' refers to indirect addressing
  - (recall that Intel uses `[']`).
Motorola 68000 µP – Assembler Examples

• move.b #$F5, d1
  – Store immediate (#) hex ($) byte (.b), $F5 into destination d1. 8-bit transfer.

• move.w (a2), d1
  – Stores contents of memory addressed by a2 into data register d1. 16-bit transfer.

• add.l d4, d5
  – Add 32-bit contents of register d4 to d5 and store the results in d5. Set flags.
## Motorola 68000 µP – Addressing Modes

- There are 14 different addressing modes (more with the 68020!)

<table>
<thead>
<tr>
<th>Mode</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data reg direct</td>
<td>$d_n$, $n = 0..7$</td>
</tr>
<tr>
<td>Addr reg direct</td>
<td>$a_n$, $n = 0..7$</td>
</tr>
<tr>
<td>Addr reg indirect</td>
<td>$(a_n)$</td>
</tr>
<tr>
<td>with Postincrement</td>
<td>$(a_n) +$</td>
</tr>
<tr>
<td>with Predecrement</td>
<td>$-(a_n)$</td>
</tr>
<tr>
<td>with Displacement</td>
<td>$d_{16}(a_n)$</td>
</tr>
<tr>
<td>with Index</td>
<td>$d_{8}(a_n, X_m)$ ($X_m$ is any $a_m$ or $d_m$)</td>
</tr>
<tr>
<td>Relative with offset</td>
<td>$d_{16}(PC)$</td>
</tr>
<tr>
<td>Relative with index and offset</td>
<td>$d_{8}(PC, X_n)$</td>
</tr>
<tr>
<td>Absolute short</td>
<td>&lt; ... &gt; (16-bits sign-extended to 32)</td>
</tr>
<tr>
<td></td>
<td>(for 000000-007FFF or FF8000-FFFFFF)</td>
</tr>
<tr>
<td>Absolute long</td>
<td>&lt; ... &gt; (32-bits)</td>
</tr>
<tr>
<td>Immediate</td>
<td>#&lt; ... &gt;</td>
</tr>
<tr>
<td>Quick immediate</td>
<td>#&lt; ... &gt; (1 byte, sign-extend to 32)</td>
</tr>
<tr>
<td>Implied</td>
<td>Register specified as part of mnemonic</td>
</tr>
</tbody>
</table>
Motorola 68000 µP – Addressing Mode Examples

• move.b #$6f,d0
  – Immediate

• move.w d3,d4
  – data reg direct. Lower 16 bits of d3 are copied to low 16-bits of d4, upper d4 not changed

• movea.l a5,a2
  – address reg direct. Size must be .w or .l; .w implies sign extension

• move.b (a0),d7
  – address register indirect. Byte pointed at by a0 copied to d7

• move.w (a5)+,d2
  – post-increment. Word pointed at by a5 copied to d2, a5 then incremented by two (.w)

• move.b -(a2),d4
  – pre-decrement. a2 decremented by one, then byte pointed at by a2 copied to d4
Motorola 68000 μP – Addressing Mode Examples

- `move.w $100(a0),d0`
  - *addr reg indirect with displacement.* Contents of memory at $a0+100_{16}$ copied to $d0$

- `move.b $08(a0,d1.w),d0`
  - *addr reg indir with index.* Note: can specify size here! Uses $b_{15}-b_0$ of $d1$ only ($addr=a0+d1.w+$08)

- `move.b $9AE0,d1`
  - *absolute short.* Sign extend to get data from address $FF9AE0$.

- `move.b $2E0000,d4`
  - *Absolute long*

- `moveq #$2C,d3`
  - *Quick Immediate.* Byte only (data encoded within instruction word). Byte is sign-extended to 32 bits.
Example: A sample assembler subroutine for the 68000:
Total: Find the sum of 16-bytes stored in memory.

```assembly
org $8000 ;load program counter

total clr.w d0 ;clear D0.
move.b #16,d1 ;initialize counter
movea.l #data,a0 ;init pointer to data

loop add.b (a0)+,d0 ;add byte, increment address
subq.b #1,d1 ;decrement counter
bne loop ;test for zero, branch not equal.
movea.l #sum,a1 ;load address to store result
move.w d0,(a1) ;store sum at sum

rts ;return from subroutine.

sum dc.w 0 ;save room for result.
data ds.b 16 ;save room for 16 data bytes.
end
```

Note:
- `dc.w` - define a constant word, operand specifies the value to be written.
- `ds.b` - define storage byte, operand specifies number of bytes, but not the contents.
Motorola 68000 µP – Stack

- Address register a7 is used to point to the stack.
- There are no push or pop instructions – (except for pea - push effective address).
- A push is done with:
  \[
  \text{move.l d3, -(a7)}
  \]
  1. Decrement a7 by four,
  2. Write 32 bits to stack

- Actually implemented as:
  1. Decrement a7 by two.
  2. Write low word of d3
  3. Decrement a7 by two.
  4. Write high word of d3
Motorola 68000 µP – Stack

• A pop is done with
  \[ \text{move.l (a7)+, d3} \]
• Stack grows down (towards lower addresses)

• \text{pea} - Push Effective Address.
  \[ \text{pea $40(a5)} \]
  – Effective address is sum of a5 and $40.
  – Result is pushed.

• \text{jsr, rts} - Subroutine calls
  – Push/pop program counter and branch.
### TABLE 2.1 68000 instruction set

<table>
<thead>
<tr>
<th>Data transfer group</th>
<th></th>
<th>Shift and rotate group</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXG</td>
<td>Exchange registers</td>
<td>ASL</td>
</tr>
<tr>
<td>LEA</td>
<td>Load effective address</td>
<td>Arithmetic shift left</td>
</tr>
<tr>
<td>LINK</td>
<td>Link and allocate</td>
<td>ASR</td>
</tr>
<tr>
<td>MOVE</td>
<td>Move data</td>
<td>LSL</td>
</tr>
<tr>
<td>MOVEA</td>
<td>Move address</td>
<td>LSR</td>
</tr>
<tr>
<td>MOVEM</td>
<td>Move multiple registers</td>
<td>ROL</td>
</tr>
<tr>
<td>MOVEQ</td>
<td>Move peripheral data</td>
<td>ROR</td>
</tr>
<tr>
<td>PEA</td>
<td>Push effective address</td>
<td>ROX</td>
</tr>
<tr>
<td>SWAP</td>
<td>Swap register halves</td>
<td>ROXL</td>
</tr>
<tr>
<td>UNLK</td>
<td>Unlink</td>
<td>ROXR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arithmetic group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Add binary</td>
<td></td>
</tr>
<tr>
<td>ADDA</td>
<td>Add address</td>
<td></td>
</tr>
<tr>
<td>ADDI</td>
<td>Add immediate</td>
<td></td>
</tr>
<tr>
<td>ADDQ</td>
<td>Add quick</td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>Clear operand</td>
<td></td>
</tr>
<tr>
<td>CMP</td>
<td>Compare</td>
<td></td>
</tr>
<tr>
<td>CMPA</td>
<td>Compare address</td>
<td></td>
</tr>
<tr>
<td>CMPi</td>
<td>Compare immediate</td>
<td></td>
</tr>
<tr>
<td>CMPM</td>
<td>Compare memory</td>
<td></td>
</tr>
<tr>
<td>DIVS</td>
<td>Divide signed numbers</td>
<td></td>
</tr>
<tr>
<td>DIVU</td>
<td>Divide unsigned numbers</td>
<td></td>
</tr>
<tr>
<td>EXT</td>
<td>Extend sign</td>
<td></td>
</tr>
<tr>
<td>MULS</td>
<td>Multiply signed numbers</td>
<td></td>
</tr>
<tr>
<td>MULU</td>
<td>Multiply unsigned numbers</td>
<td></td>
</tr>
<tr>
<td>NEG</td>
<td>Negate</td>
<td></td>
</tr>
<tr>
<td>NEGX</td>
<td>Negate with extend</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>Subtract binary</td>
<td></td>
</tr>
<tr>
<td>SUBA</td>
<td>Subtract address</td>
<td></td>
</tr>
<tr>
<td>SUBi</td>
<td>Subtract immediate</td>
<td></td>
</tr>
<tr>
<td>SUBQ</td>
<td>Subtract quick</td>
<td></td>
</tr>
<tr>
<td>SUBX</td>
<td>Subtract with extend</td>
<td></td>
</tr>
<tr>
<td>TAS</td>
<td>Test and set</td>
<td></td>
</tr>
<tr>
<td>TST</td>
<td>Test</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logical group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Logical AND</td>
<td></td>
</tr>
<tr>
<td>ANDI</td>
<td>AND immediate</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR</td>
<td></td>
</tr>
<tr>
<td>ORI</td>
<td>OR immediate</td>
<td></td>
</tr>
<tr>
<td>EOR</td>
<td>Exclusive OR</td>
<td></td>
</tr>
<tr>
<td>EORI</td>
<td>Exclusive OR immediate</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>Logical complement</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit manipulation group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BCHG</td>
<td>Bit change</td>
<td></td>
</tr>
<tr>
<td>BCLR</td>
<td>Bit clear</td>
<td></td>
</tr>
<tr>
<td>BSET</td>
<td>Bit set</td>
<td></td>
</tr>
<tr>
<td>BTST</td>
<td>Bit test</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary coded decimal group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>Add BCD</td>
<td></td>
</tr>
<tr>
<td>NBCD</td>
<td>Negate BCD</td>
<td></td>
</tr>
<tr>
<td>SBCD</td>
<td>Subtract BCD</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program control group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bcc*</td>
<td>Conditional branch</td>
<td></td>
</tr>
<tr>
<td>DBcc*</td>
<td>Decrement and branch</td>
<td></td>
</tr>
<tr>
<td>Sccc*</td>
<td>Conditional set</td>
<td></td>
</tr>
<tr>
<td>BRA</td>
<td>Branch always</td>
<td></td>
</tr>
<tr>
<td>BSR</td>
<td>Branch to subroutine</td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
<td></td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to subroutine</td>
<td></td>
</tr>
<tr>
<td>RTR</td>
<td>Return and restore</td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td>Return from subroutine</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System control group</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ANDI SR</td>
<td>AND immediate to SR</td>
<td></td>
</tr>
<tr>
<td>EORI SR</td>
<td>EOR immediate to SR</td>
<td></td>
</tr>
<tr>
<td>MOVE SR</td>
<td>Move to/from SR</td>
<td></td>
</tr>
<tr>
<td>MOVE USP</td>
<td>Move to/from USP</td>
<td></td>
</tr>
<tr>
<td>ORI SR</td>
<td>OR immediate to SR</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>Reset processor</td>
<td></td>
</tr>
<tr>
<td>RTE</td>
<td>Return from exception</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>Stop processor</td>
<td></td>
</tr>
<tr>
<td>CHK</td>
<td>Check register</td>
<td></td>
</tr>
<tr>
<td>ILLEGAL</td>
<td>Illegal instruction</td>
<td></td>
</tr>
<tr>
<td>TRAP</td>
<td>Trap call</td>
<td></td>
</tr>
<tr>
<td>TRAPV</td>
<td>Trap on overflow</td>
<td></td>
</tr>
<tr>
<td>ANDI CCR</td>
<td>AND immediate to CCR</td>
<td></td>
</tr>
<tr>
<td>ORI CCR</td>
<td>OR immediate to CCR</td>
<td></td>
</tr>
<tr>
<td>EORI CCR</td>
<td>EOR immediate to CCR</td>
<td></td>
</tr>
<tr>
<td>MOVE CCR</td>
<td>Move to/from CCR</td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No operation</td>
<td></td>
</tr>
</tbody>
</table>

*Note: cc stands for condition code*
Motorola 68000 µP – Hardware

• 16-bit µP - 16-bit data bus \( (D_{15}-D_0) \).
  - Internal data paths are 32 bit
• 24-bit address bus.
  - \( (UDS, LDS, A_1-A_{23}) \)
• No multiplexing of busses!
• Clock speeds of 4-12.5 MHz.
  - 10/12/16/20 MHz for CMOS version (1/10th power consumption)

*Note that we will use ‘d7’ for data register d7 and ‘D7’ for data bus line D7. Likewise for a7 vs. A7.*
Motorola 68000 µP – Hardware

### Clock Circuit

![Clock Circuit Diagram]

- **D0** - 5
- **AS** - 6
- **UDS** - 7
- **LDS** - 8
- **R/W** - 9
- **DTACK** - 10
- **BG** - 11
- **BGACK** - 12
- **BR** - 13
- **VCC** - 14
- **CLK** - 15
- **GND** - 16
- **HALT** - 17
- **RESET** - 18
- **VMA** - 19
- **E** - 20
- **VPA** - 21
- **BERR** - 22
- **IPL2** - 23
- **IPL1** - 24
- **IPL0** - 25
- **FC2** - 26
- **FC1** - 27
- **FC0** - 28
- **A1** - 29
- **A2** - 30
- **A3** - 31
- **A4** - 32
- **A5** - 33
- **A6** - 34
- **A7** - 35
- **A8** - 36
- **A9** - 37
- **A10** - 38
- **A11** - 39
- **A12** - 40
- **A13** - 41
- **A14** - 42
- **A15** - 43
- **A16** - 44
- **A17** - 45
- **A18** - 46
- **A19** - 47
- **A20** - 48
- **A21** - 49
- **A22** - 50
- **A23** - 51
- **A24** - 52
- **A25** - 53
- **A26** - 54
- **A27** - 55
- **A28** - 56
- **A29** - 57
- **A30** - 58
- **A31** - 59
- **A32** - 60
- **A33** - 61
- **A34** - 62
- **A35** - 63
- **A36** - 64

---

(a) 68000 CPU signals

**Microprocessor Systems**
Motorola 68000 µP – Asynchronous bus control

- **AS** Address strobe: valid address is on address bus.
- **R/W**: for read, 0 for write.
- **UDS**: Upper data strobe. Data on D_{15}-D_8 (like BHE).
- **LDS**: Lower data strobe. Data on D_7-D_0 (like BLE).
- **DTACK**: Data transfer acknowledge.
  - Signal by external hardware that µP may complete the current bus cycle.
  - During read, µP latches data when DTACK = 0.
  - During write, µP puts data on bus and keeps it there until DTACK = 0.
Motorola 68000 µP – Asynchronous bus control

FIGURE 7.12 Decoding memory read/write signals
Motorola 68000 µP – Hardware

• System control
  - **RESET**: reset the µP. (in)
  - **HALT**: µP puts the busses into high-impedance state
    - (Equivalent to HOLD on 8086) (in/out).
  - **BERR**: Bus error - illegal memory location (in)
    - YOU must generate this if DTACK or VPA never returns

![Processor HALT timing diagram](image)
Motorola 68000 µP – Hardware

• $FC_0, FC_1, FC_2$:
  - Encoded processor states.
  - only valid with $AS = 0$ (address strobe).
  - $FC_0FC_1FC_2 = 111$: interrupt acknowledge.

<table>
<thead>
<tr>
<th>TABLE 7.1</th>
<th>Function code outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FC_2$</td>
<td>$FC_1$</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*By Motorola, for future use.

• $IPL_0, IPL_1, IPL_2$
  - Encoded interrupt priority level.
  - Seven interrupt levels.
  - Level 7 (all zeros) is highest
Motorola 68000 µP – Bus arbitration

- Bus arbitration control:
  - BR: Bus request (in)
  - BG: Bus grant (out)
  - BGACK: Bus grant acknowledgment (in)

- Used to place 68000 busses in high impedance state so that a peripheral can use the bus.

- Sequence:
  1. External device sets BR = 0.
  2. 68000 sets BG = 0.
  3. External device waits for BG = 0, AS = 1, DTACK = 1, BGACK = 1, then will set BGACK = 0 to take control of busses.
Motorola 68000 µP – Bus arbitration

FIGURE 7.11  Bus arbitration logic block diagram
Motorola 68000 µP – Hardware

• Interface to 6800 peripherals:
  - E: Clock (out)
  - VPA: Valid Peripheral address (in).
    - Should be asserted by interface circuitry whenever a 6800 peripheral has been selected.
  - VMA: Valid Memory address (out).
    - Asserted by µP when internal clock is in synch with E-clock. Connected to second peripheral CS pin.
### TABLE 7.3 Summary of 68000 signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Input</th>
<th>Output</th>
<th>Tristate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FC₀–FC₂</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>VMA</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>VPA</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>BERR</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>HALT</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>IPL₀–IPL₂</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BG</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>BGACK</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>AS</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R/W</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>UDS</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>LDS</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>DTACK</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>A₁–A₂₃</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>D₀–D₁₅</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7.13** Buffering the address and data buses

Buffered bidirectional data bus

UDS and LDS make up the “24th” address line

Buffered unidirectional address bus
Motorola 68000 µP – Read/Write Bus Cycles

- The 68000µP uses $\overline{AS}=0$ to signal a memory access.
- When memory sees $\overline{AS}=0$, and it is the correct address, it responds by pulling DTACK low which tells the µP to proceed with the data transfer.
- Bus cycles are divided into a minimum of eight states, S0 - S7. Each state is 1/2 a clock cycle.
Motorola 68000 µP – Read Cycle

- **S0**: Address bus is in high impedance state, R/W is set to 1 (read operation).
- **S1**: Valid address appears on address bus.
- **S2**: AS goes low (valid address), LDS and UDS are set to the desired state.
- **S3, S4**: Minimum time given to memory to signal with DTACK=0.
- **S5**: µP looks for DTACK=0.
  - if DTACK=1, insert two wait states then test DTACK again.
  - if DTACK=0, continue with S6 and S7.
- **S6**: Nothing new happens.
- **S7**: Latch data into µP, set AS, UDS, and LDS to 1. Memory releases DTACK
Motorola 68000 μP – Read Cycle

- **CLK**
- **A1-A23**
- **AS**
- **UDS**
- **LDS**
- **R/W**
- **DTACK**
- **D0-D7**
- **FC0-FC2**

**68000 Read Cycle**

- S0
- S1
- S2
- S3
- S4
- S5
- S6
- S7

- **ASYNCHRONOUS!**
  - From memory device.
- **CHECK DTACK**
  - (S5)
  - DTACK released when AS released
  - READ DATA
  - (end of S6)

- **Data may not be avail when DTACK drops**
- **Only guaranteeing that it WILL be ready in time**
  - (end of S6)
Motorola 68000 µP – Write Cycle

- **S0**: Address bus is in high impedance state.
- **S1**: Valid address appears on address bus.
- **S2**: AS goes low (valid address), R/W is set to 0 (write operation). (LDS and UDS are delayed to allow the bus transceivers to switch direction, and to allow the memory time to prepare.)
- **S3**: Valid data is placed on the bus by the µP.
- **S4**: LDS and UDS are set to the desired state.
- **S5**: µP looks for DTACK=0.
  - if DTACK=1, insert two wait states then test DTACK again.
  - if DTACK=0, continue with S6 and S7.
- **S6**: Nothing new happens.
- **S7**: Set AS, UDS, and LDS to 1. Memory releases DTACK
Motorola 68000 µP – Write Cycle

ASYNCHRONOUS!
From memory, can arrive any time before S5 without causing wait states
Motorola 68000 µP – Memory Organization

- 68000 is byte-addressable.
- 16-bit words and 32-bit long words must begin at an even address, otherwise address error.
- 68000 is a big-endian processor:
  - 16-bit words are stored with the lower-order byte (endian) in the higher-order (big) memory address.
  - (Recall that the 8086 is little-endian)
- Consequences:

<table>
<thead>
<tr>
<th>Bank</th>
<th>8086</th>
<th>68000</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D_7-D_0)</td>
<td>Even (BLE)</td>
<td>Odd (LDS)</td>
</tr>
<tr>
<td>(D_{15}-D_8)</td>
<td>Odd (BHE)</td>
<td>Even (UDS)</td>
</tr>
</tbody>
</table>
Motorola 68000 \( \mu \)P – Memory Organization

- 8086 memory is usually drawn with *odd* bank on left, and *even* bank on right.

<table>
<thead>
<tr>
<th></th>
<th>Odd</th>
<th>Even</th>
</tr>
</thead>
<tbody>
<tr>
<td>000003H</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>00001H</td>
<td></td>
<td>00002H</td>
</tr>
<tr>
<td></td>
<td>( D_{15-0} ) (BHE)</td>
<td>( D_{7-0} ) (BLE)</td>
</tr>
</tbody>
</table>

- 68000 memory is usually drawn with *even* bank on left, and *odd* bank on right.

<table>
<thead>
<tr>
<th></th>
<th>Even</th>
<th>Odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>$000002$</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>$000000$</td>
<td></td>
<td>$0000003$</td>
</tr>
<tr>
<td></td>
<td>( D_{15-0} ) (UDS)</td>
<td>( D_{7-0} ) (LDS)</td>
</tr>
</tbody>
</table>

SYSC3601 32 Microprocessor Systems
### Motorola 68000 μP – Memory Organization

- **Ex 68000 memory segment:**

<table>
<thead>
<tr>
<th></th>
<th>Even (High)</th>
<th>Odd (Low)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00100A</td>
<td>C3</td>
<td>8F</td>
</tr>
<tr>
<td>$001008</td>
<td>3F</td>
<td>6B</td>
</tr>
<tr>
<td>$001006</td>
<td>00</td>
<td>4A</td>
</tr>
<tr>
<td>$001004</td>
<td>23</td>
<td>08</td>
</tr>
</tbody>
</table>

- Byte@$1004 = $23 (high half)
- Byte@$1005 = $08 (low half)
- Word@$1006 = $004A (both halves)
- Word@$1008 = $3F6B (both halves)
- Word@$1009 = Address Error
- Long word@$1008 = $3F6BC38F (both halves, twice)
- Long word@$1005 = Address Error
Motorola 68000 µP – Memory Interfacing

• Almost identical to the 8086 except:
  1. Switch even and odd banks
  2. Must generate DTACK
  3. Must use AS, R/W, UDS and LDS for control.

• During a byte-read operation, the µP will select the correct half of the data bus depending on whether it's an even or odd address (similar to 8086). (However, most designs provide separate read strobes for even and odd banks.)

• Separate write strobes are required for even and odd banks so that data is not written to the wrong memory bank.
Motorola 68000 µP – Memory Interfacing

• Block diagram of DTACK circuit:

• DTACK delay generator:

‘Device select’ signal from address decoder. (goes low end of S1)

Goes low on rising edge of S4

Goes low on rising edge of S6 (too late, causes 1 clock cycle delay)

‘PRE’ presets the flip-flop (sets it to 1)
Motorola 68000 µP – I/O Interfacing

- All I/O is memory-mapped.
- Decoding is the same as for memory.
- One still must generate DTACK.

Would require another buffer/latch pair for UDS for a 16-bit I/O interface. (connected to D15-D8).

**FIGURE 9.1** Memory-mapped I/O circuitry
The 68000 has three execution states:

1. Normal - running user program.
2. Halted - not executing instructions. (perhaps because of a system failure such as a double bus fault, or due to HALT pin).
3. Exception (processing) state - includes interrupts, but goes beyond the usual notion of interrupts.
Motorola 68000 µP – Exceptions (Interrupts)

• Two privilege states.
  – User and Supervisor.
  – Some instructions are only available in supervisor state.
    • STOP, RESET, RTE, MOVE/AND/EOR/OR to SR, MOVE to USP
  – Separate stack pointers.
  – Provides security for operating systems etc.
  – All exception processing is done in supervisor state.
  – The only way to get to supervisor state is through an exception (or reset).
Motorola 68000 µP – Exceptions (Interrupts)

- Can use privilege state for memory management
  - i.e. include FC2-pin in memory interface.
Motorola 68000 µP – Exceptions (Interrupts)

- **Interrupt Vectors and the vector table**
  - A vector number is one byte (0-255)
  - Vector Table:
    - Occupies the first 1kbyte (0000 - 3FF) of memory.
    - Each entry (except the first) is a 32-bit address pointing to the start of the exception handler code.

- **Example**: Vector = $5
  (Divide by zero)

<table>
<thead>
<tr>
<th>Byte 3</th>
<th>Byte 2</th>
<th>Byte 1</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$000</td>
<td>$001</td>
<td>$002</td>
<td>$003</td>
</tr>
<tr>
<td>$014</td>
<td>$015</td>
<td>$016</td>
<td>$017</td>
</tr>
<tr>
<td>$3FF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

68000 fetches new PC
From address $5 \times 4 = $014

- **BIG Endian!!**
## Motorola 68000 µP – Exceptions (Interrupts)

**TABLE 4.1 Exception vector assignments**

<table>
<thead>
<tr>
<th>Vector Numbers</th>
<th>Dec</th>
<th>Hex</th>
<th>Space</th>
<th>Assignment</th>
</tr>
</thead>
</table>
| 0              | 0   | 000 | SP    | Reset: Initial SS  
| 2              | 4   | 004 | SP    | Reset: Initial PC  
| 3              | 8   | 008 | SD    | Bus error  
| 4              | 12  | 00C | SD    | Address error  
| 5              | 16  | 010 | SD    | Illegal instruction  
| 6              | 20  | 014 | SD    | Zero divide  
| 7              | 24  | 018 | SD    | CHK instruction  
| 8              | 28  | 01C | SD    | TRAPV instruction  
| 9              | 32  | 020 | SD    | Privilege violation  
| 10             | 36  | 024 | SD    | Trace  
| 11             | 40  | 028 | SD    | Line 1010 emulator  
| 12             | 44  | 02C | SD    | Line 1111 emulator  
| 13             | 48  | 030 | SD    | (Unassigned, reserved)  
| 14             | 52  | 034 | SD    | (Unassigned, reserved)  
| 15             | 56  | 038 | SD    | Format error  
| 16–23          | 60  | 03C | SD    | Uninitialized interrupt vector  
|                | 64  | 040 | SD    | (Unassigned, reserved)  
|                | 92  | 05C |        |              
| 24             | 96  | 060 | SD    | Spurious interrupt  
| 25             | 100 | 064 | SD    | Level 1 interrupt autovector  
| 26             | 104 | 068 | SD    | Level 2 interrupt autovector  
| 27             | 108 | 06C | SD    | Level 3 interrupt autovector  
| 28             | 112 | 070 | SD    | Level 4 interrupt autovector  
| 29             | 116 | 074 | SD    | Level 5 interrupt autovector  
| 30             | 120 | 078 | SD    | Level 6 interrupt autovector  
| 31             | 124 | 07C | SD    | Level 7 interrupt autovector  
| 32–47          | 128 | 080 | SD    | TRAP instruction vectors  
| 48–63          | 188 | 0BC | SD    | (Unassigned, reserved)  
|                | 255 | 0FF |        |              
| 64–255         | 256 | 100 | SD    | User interrupt vectors  
|                | 1020| 3FC | | |

1. Vector numbers 12, 13, 16 through 23, and 49 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.
2. SP denotes supervisor program space, and SD denotes supervisor data space.
3. Reset vector (0) requires four words, unlike the other vectors, which only require two words, and are located in the supervisor program space.
4. MC68010 only. This vector is unassigned, reserved on the MC68000 and MC68008.
5. The spurious interrupt vector is taken when there is a bus error indication during interrupt processing.
6. TRAP #n uses vector number 32 + n.
Motorola 68000 µP – Exceptions (Interrupts)

- Exception Processing Sequence
  1. Save the status register in a temporary register and set the S-bit so that the 68000 can enter supervisor mode.
  2. Get the vector number. There are several ways:
     (a) may be determined internally by processor.
     (b) external interrupts can be “auto-vectored“ (to come).
     (c) external interrupts can provide a vector number (i.e. type) on D7-D0 during the interrupt acknowledge cycle.
Motorola 68000 µP – Exceptions (Interrupts)

3. Save processor information onto supervisor stack:
   (a) push PC low word.
   (b) push PC high word.
   (c) push status register (from saved temporary unmodified version).

4. Fetch new PC from the vector table.
5. Execute exception handler.
6. Return with RTE. Pops SR, then PC.

• Note: Exceptions can be nested – not masked automatically like 8086 does.
Motorola 68000 µP – Exceptions (Interrupts)

- 68000 Hardware Interrupts
  - Seven levels of external interrupts depending on IPL2, IPL1, and IPL0.
  - Level 0, all IPLs = 1, no interrupt.
  - Level 7, all IPLs = 0, highest priority (non-maskable).
  - Interrupt priority mask (bits 8, 9, and 10 of SR) is set to disable lower priority interrupts.

Example circuit to generate a level-7 interrupt using a single push-button.

We can develop more complex circuits to generate multiple interrupt levels depending on the source of the interrupt request.
Motorola 68000 µP – Exceptions (Interrupts)

• Interrupt Acknowledge Cycle
  – (asynchronous, hardware interrupt requests)
  1. Device and interrupt logic set $\text{IPL2, IPL1, IPL0}$.
  2. $\mu$P completes current instruction.
  3. $\mu$P enters interrupt acknowledge cycle.
    (a) $\text{FC2, FC1, FC0} = 111$.
    (b) $\overline{\text{AS}} = 0$, $\overline{\text{LDS}} = 0$, $\overline{\text{R/W}} = 1$.
      $A_3, A_2, A_1 = \text{requested interrupt level}.$
• Interrupt Acknowledge Cycle con’t

4. External logic may do one of two things:
   
   (a) Supply a vector number.
      
      – Place 8-bit vector number of $D_7-D_0$.
      
      – Pull DTACK low.
      
      – μP will read $D_7-D_0$.

   (b) Request an “auto-vector”.

      – Pull VPA low. Leave DTACK high.
      
      – μP generates its own vector based on interrupt level first supplied to IPL inputs.
      
      – Autovectors point to locations $064$ through $07F$ in vector table.

      – Autovectors should be used whenever 7 or less interrupt types are needed.

5. Proceed with exception handling steps from slide 42
The response to an interrupt is quite lengthy and complex:

1. Resolve priorities from external interrupt request, present appropriate 3-bit code on IPL\(_{2:0}\).
2. Monitor FC\(_{2:0}\) for intr acknowledge cycle.
   - AS=0,
   - R/W=1,
   - LDS=0
   - A\(_{3:1}\) = requested interrupt level
3. Either:
   3a) provide vector number on D\(_{7:0}\) and pull DTACK low,
   OR
   3b) request autovector by pulling VPA low.
**Motorola 68000 μP – Peripheral chips**

### 68681 DUART
- Contains 2 UARTs, independently programmable
  - Both channels can provide simultaneous Tx/Rx.
- Interface using internal control/data/status registers selected via RS\(_{4-1}\) pins.
- Also provides 6 parallel inputs and 8 parallel outputs
  - Can be used for handshaking signals or standard I/O pins.

![Diagram of 68681 DUART](image)

**FIGURE 10.17** Using the 68681 in a 68000-based system
Motorola 68000 µP – Peripheral chips

68230 Parallel Interface/Timer (PI/T)
- Contains 3 8-bit parallel ports
  - Can be input, output, bidirectional
  - Ports A&B can form 16-bit port
- Also contains an internal 24 bit timer
  - Count down, square wave generator, etc.
- 23 internal data/control/status registers selected via RS_{5-1} pins.
- H_{4-1} pins are handshaking for ports A&B
  - Can cause interrupts
- PortC can be used as general I/O pins, interrupt request/acknowledge, timer inputs/outputs, or DMA request.

FIGURE 10.25 Interfacing the 68230 PI/T
Keyboard scanning with a 68230 PI/T

- Drive PA\textsubscript{3-0} in sequence
- Read PB\textsubscript{3-0} to check for key depressed

![Sixteen-key keypad scanner using the 68230](image)

**TABLE 10.1** Keyboard scanning codes

<table>
<thead>
<tr>
<th>Parallel outputs</th>
<th>Buttons scanned</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA\textsubscript{3} PA\textsubscript{2} PA\textsubscript{1} PA\textsubscript{0}</td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>3, 2, 1, 0</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>7, 6, 5, 4</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>B, A, 9, 8</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>F, E, D, C</td>
</tr>
</tbody>
</table>
Motorola 68000 µP – Peripheral chips

4 Digit Display with a 68230 PI/T
- Select segments using PA_{7-0}
- Select display chip using PB_{3-0}
  - Have to scan through chips quickly to avoid flicker effect.
- Use timer to cause interrupts to cycle through chips.
**Motorola 68000 µP – Peripheral chips**

**68881 Math co-processor**
- Similar to 8087
- 8 Internal 80-bit floating point registers
- 40 floating point instructions
- 32-bit data bus
  - Optimally used with 68020 (32-bit bus)
- $A_0$ and SIZE are used to configure the 68881 for the size of the µP’s data bus.

**FIGURE 10.39** Floating-point coprocessor connections to the 68000
Motorola 68000 µP – Peripheral chips

INTEL 8279 Keyboard/Display Chip
- For non-68000 series chips, must generate DTACK signal using interface circuitry.
- Must also create separate RD and WR signals from joint R/W

Other peripheral chips:
- 68153 BUS Interrupt Module
- 68440 Dual DMA Controller
- 6851 Memory Management Unit
- 68901 Multifunction Peripheral
- 68465 Floppy Disk Controller
- 68452 Bus Arbitration Module
- 68590 LAN Controller for Ethernet
- 68652 Multiprotocol Communications Controller
- 68824 Token-Passing Bus Controller
- 68486/68487 Raster Memory System
- 68184 Broadband Interface Controller

FIGURE 10.41 Interfacing the 8279 to the 68000