Micro-Baby

Micro-Baby is a simple computer architecture, in fact, very simple. All microcontrollers and microprocessors are computer architectures, in most cases fairly simple ones. However, in today’s world even microcontrollers and microprocessors are eons beyond basic. They include many advanced features which often obscure the basic concepts of a stored program computer.

It is assumed that the reader of this book and this chapter possesses a basic understanding of the binary number system and the implementation of logic equations in digital logic using AND, OR, NAND, NOR, and NOT gates. If you wish to brush up a quick review of these topics is provided in Appendix A and Appendix B.

Micro-Baby is an accumulator based load-store architecture. In fact, that is a good place to start this chapter. The basic concepts of a stored program computer will be presented and how it can be implemented with this architecture. Just like the architecture of a home, the architecture of a computer is the ‘how’ the pieces are put together. Building on this concept the instructions of Micro-Baby will be presented, the stored program that allows Micro-Baby to do something useful.

A. Accumulator Based Load-Store Architecture



The Micro Baby implementation of this has the following structure:



MircoBaby instructions:

Instructions are 2 bytes in length. The first byte is the op-code and the second where in data memory the 2nd operand is located. The first operand is in the accumulator and that is where the result will also be.

Arithmetic: (0100 0xxx)

ADD Add value to accumulator

ADDC Add value and carry to accumulator

SUB Subtract value from accumulator

SUBC Subtract value and borrow from accumulator

INC Increment accumulator

DEC Decrement accumulator

Logical (0101 xxxx)

AND Logically AND the accumulator and value

OR Logically OR the accumulator and value

INV Logically invert the accumulator

XOR Logically XOR the accumulator and value

CLRA Clear Accumulator

SLA *not sure yet*

SRA *not sure yet*

Jumps: (64) (starts 11xx xyyy) – bit x indicates used or not, y indicates clear or set

Carry clear or set

Zero clear or set

Negative clear or set

Jump always – when not using any of the conditions (11000xxx)

Tests (0100 1xxx)

CMP Compare (executes Accumulator – argument) does not modify accumulator.

Data Movement (10y x xxmm) - 2 byte instruction – 2nd byte is address

mm is addressing mode

LDA Load Accumulator (101x xxxmm)

STA Store Accumulator (100x xxxmm)

ADDRESSING MODES

Immediate

Direct

Indirect *Not sure yet*

**Memory**

Data divided into data memory and instruction memory

Each is 256 bytes – addressable by 8 bits

CPU to memory signals – 8-bit data bus, 8-bit address bus, r/w signal, and timing with is used to generate and enable signal. Data is placed on the bus when enable is valid. Data, address, and r/w go to high impedance when not needed by bus master.

**mb Controller**

The controller contains the Program counter, the instruction register, and the status register.

The main part of the controller is a state machine which alternates between two main states, Fetch and Execute. Within each of these there are multiple subcycles.

The first cycle completes the action MEM(PC) 🡪 IR

PC + 1 🡪 PC

The second cycle then performs the action of the instruction generating the control signals for the datapath to execute the instruction. The control signals during the execute cycle are determined by the dpcsvec, , a 0 to 6 vector that specifies the control signals of the data path during the execute cycle. They are detailed later in this document.

If the instruction is inherent, like Invert the content of the Accumulator, or immediate, then only one execute cycle is needed. If the addressing mode is direct, then a 2nd execute cycle is needed. During the 1st execute cycle (2nd cycle of execution) the address of the operand which is the 2nd byte of the instruction is fetched from instruction memory and latched in the temp\_addr\_register in the controller. In the 2nd execute cycle (3rd cycle overall) the argument is fetched or stored in data memory.

Dpcsvec pos signal value meaning value meaning

0 Aal 0 Mux out is Dbus 1 Mux output is allures

1 Bbu 0 Mux out is Dbus 1 Zero

2 Ldacc 0 no action 1/\ Rising edge ld accum

3 Dracc 0 Don’t drive 1 drive accum on Dbus

4 Csel 00 Cin

01 NOT Cin

10 ‘0’

11 ‘1’

6 AddSub 0 Input NOT B 1 Input B

7 Arlo 0 Logic unit output 1 ALU output

**HDL model**

The memory module is specified as follows:

Each memory block (data or instruction) has the following features as indicated by the diagram.



The bus timing looks as follows: (the control signals need to be added.



The ALU has the following structure and control signals.



The datapath has the following structure and control signals



The controller has the following structure.







PC Unit

**Assignment VP2:**

1. Verify the Microbaby structures available. The one structure that is not yet complete is the controller. Work on completing its design and verification after completing verification of the components of the architecture.

2. Write a testplan to test and verify the components of the architecture. Testplan is due the Monday December 1st.

3. Work on design of the controller. A microcoded controller is desired. There is high probability a behavioral controller will be available from the design group soon.

4. Write a final verification report that details your outcomes on these tasks.

INSTRUCTION CODING

|  |  |  |  |
| --- | --- | --- | --- |
| **I NSTR** | **Op Code** | **2nd byte** | **Addr mode comment** |
| LDA | 1000 0010 | Addr | *Direct* |
| LDA | 1000 0001 | Argument | *Immediate* |
| STA | 1010 0010 | Addr | *Direct* |
| ADD | 0100 0010 | Addr | Direct |
| ADD | 0100 0001 | Argument | Immediate |
| ADDC | 0100 1010 | Addr | Direct |
| ADDC | 0100 1001 | Argument | Immediate |
| SUB | 0100 0010 | Addr | Direct |
| SUB | 0100 0001 | Argument | Immeddiate |
| SUBC | 0101 1010 | Addr | Direct |
| SUBC | 0101 1001 | Argument | Immeddiate |
| INC | 0100 1100 | Na | Inherent |
| DEC | 0100 0100 | NA | Inherent |
| AND | 0101 1010 | Addr | Direct |
| AND | 0101 1001 | Argument | Immeddiate |
| OR | 0101 1110 | Addr | Direct |
| OR | 0101 1101 | Argument | Immeddiate |
| INV | 0101 0000 | Na | Inherent |
| XOR | 0101 0010 | Addr | Direct |
| XOR | 0101 0001 | Argument | Immeddiate |
| CLRA | 0101 1111 | Na | Inherent |
| CMP | 0100 1010 | Addr | Direct |
| CMP | 0100 1001 | Argument | Immeddiate |
| JMP | 11cn zuuu | Addr | Direct |

Within the controller there are internal control signals for the Program Counter Unit

These allow for selection of a fixed value or the PC+1 value to be loaded. This is PCsel signal.

There is also the ldnewpc signal and on the rising edge of this signal the muxout value is latched into the PC.

During the instruction decode phase a SIGNAL dpcs (for datapath control signals) is given values along with SIGNAL dpfun which is the alu function. These are used for control in the next cycle.

Control Vector Interpretation

dpFunction vector

value Alu operation

Memcsvec – the interpretation of the 2 bits of the vector

Bit

3 1=inherent – only a 1 byte instrction

2 1=direct addressing mode – instruction will have 3rd cycle

1 msb the value of the rw control signal 0=memory drives data on bus

1=memory stores data on bus

0 lsb which memory gets activated 0 – instruction memory 1 – data memory

Dpcsvec pos signal value meaning value meaning

0 Aal 0 Mux out is Dbus 1 Mux output is allures

1 Bbu 0 Mux out is Dbus 1 Zero

2 Ldacc 0 no action 1/\ Rising edge ld accum

3 Dracc 0 Don’t drive 1 drive accum on Dbus

4/5 Csel 00 Cin

01 NOT Cin

10 ‘0’

11 ‘1’

6 AddSub 0 Input NOT B 1 Input B

7 Arlo 0 Logic unit output 1 ALU output

8 Bmltch 0 no action /\1 Rising edge ld b input

For a LDA immediate mode the value of dpcs is

Pos CtlSig value

0 Aal 0 Load from Data bus

1 Bbu 1 select fixed val

2 Ldacc 1 should be loaded

3 Dracc 0 Don’t drive

4 Csel 00 Carry selection signal for ALU

5 AddSub 0 Add or Subtract

6 Arlo 0 Arithmetic or Logic

Arithmetic: (0100 0xxx)

ADD Add value to accumulator

ADDC Add value and carry to accumulator

SUB Subtract value from accumulator

SUBC Subtract value and borrow from accumulator

INC Increment accumulator

DEC Decrement accumulator

Logical (0101 xxxx)

AND Logically AND the accumulator and value

OR Logically OR the accumulator and value

INV Logically invert the accumulator

XOR Logically XOR the accumulator and value

CLRA Clear Accumulator

SLA *not sure yet*

SRA *not sure yet*

Jumps: (64) (starts 11xx xyyy) – bit x indicates used or not, y indicates clear or set

Carry clear or set

Zero clear or set

Negative clear or set

Jump always – when not using any of the conditions (11000xxx)

Tests (0100 1xxx)

CMP Compare (executes Accumulator – argument) does not modify accumulator.

Data Movement (10y x xxmm) - 2 byte instruction – 2nd byte is address

mm is addressing mode

LDA Load Accumulator (101x xxxmm)

STA Store Accumulator (100x xxxmm)

ADDRESSING MODES

Immediate

Direct

Indirect *Not sure yet*

Control Signal Modification

2 control signal vectors for each of the possible states

e\_csvec

1 R/W 1=memory stores bus 0=memory drives bus

2 Direct Addressing 1=yes 0=no

3 Immediate Addr 1=yes 0=no

4 Inherent Mode 1=yes 0=no

5 incr PC at end of cyc 1=yes 0=no

6 Latch flags from dp 1=yes 0=no

7 Clear the carry flag 1=yes 0=no

8 Set the carry flag 1=yes 0=no

e2\_csvec

1 R/W 1=memory dirves bus 0=memory stores bus

Same def as ecsvec

Instruction successfully implemented.

LDA immediate 55

LDA immediate AA

LDA direct from address 10 which has $F0

STA direct at address 11

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I NSTR** | **Op Code** | **2nd by** | **Addr mode** | **dpcsvec** | **ecsvec/e2csvec** |
| ***LDA*** | 1000 0010 | Addr | *Direct* | 00000000 | 01001 / 01000 |
| ***LDA*** | 1000 0001 | Arg | *Immediate* | 01100000 | 00101 |
| ***STA*** | 1010 0010 | Addr | *Direct* |  |  |
| ADD | 0100 0010 | Addr | Direct |  |  |
| ***ADD*** | 0100 0001 | Arg | *Immediate* |  |  |
| ADDC | 0100 1010 | Addr | Direct |  |  |
| **ADDC** | 0100 1001 | Arg | *Immediate* |  |  |
| SUB | 0101 0010 | Addr | Direct |  |  |
| ***SUB*** | 0101 0001 | Arg | *Immeddiate* |  |  |
| SUBC | 0111 0010 | Addr | Direct |  |  |
| **SUBC** | 0111 0001 | Arg | **Immeddiate** |  |  |
| INC | 0100 1100 | Na | Inherent |  |  |
| DEC | 0100 0100 | NA | Inherent |  |  |
| AND | 0101 1010 | Addr | Direct |  |  |
| ***AND*** | 0101 1001 | Arg | *Immeddiate* |  |  |
| OR | 0101 1110 | Addr | Direct |  |  |
| ***OR*** | 0101 1101 | Arg | *Immeddiate* |  |  |
| INV | 0101 1000 | Na | Inherent |  |  |
| XOR | 0101 0110 | Addr | Direct |  |  |
| ***XOR*** | 0101 0101 | Arg | *Immeddiate* |  |  |
| CLRA | 0100 1111 | Na | Inherent |  |  |
| **CLRC** | 0100 0000 | Na | *Inherent* |  |  |
| **CSET** | 0100 1000 | Na | *Inherent* |  |  |
| CMP | 0100 1010 | Addr | Direct |  |  |
| CMP | 0100 1001 | Arg | Immeddiate |  |  |
| JMP | 11cn zuuu | Addr | Direct |  |  |

The tests in initial file

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| L | Pneumonic |  | Op Code $ | Arg $ | ALU out | Data mem |
| 00 | LDA #$AA |  | 81 | AA |  |  |
| 02 | LDA #$55 |  | 81 | 55 |  |  |
| 04 | LDA $10 |  | 82 | 10 |  | 10= |
| 06 | LDA $11 |  | 82 | 11 |  | 11= |
| 08 | STA $30 |  | A2 | 30 |  |  |
| 0A | LDA $30 |  | 82 | 30 |  |  |
| 0C | ADD #01 |  | 41 | 01 |  |  |
| 0E | ADD #$10 |  | 41 | 10 |  |  |
| 10 | SUB #$01 |  | 51 | 01 |  |  |
| 12 | SUB #$11 |  | 51 | 11 |  |  |
| 14 | LDA #$AA |  | 81 | AA |  |  |
| 16 | AND #$0F |  | 59 | 0F | 0A |  |
| 18 | AND #$0F |  | 59 | 0F | 0A |  |
| 1A | AND #$FF |  | 59 | FF | 0A |  |
| 1C | AND #$F0 |  | 59 | F0 | 00 |  |
| 1E | LDA #$AA |  | 81 | AA |  |  |
| 20 | AND #$5B |  | 59 | AB |  |  |
| 22 | OR #$FF |  | 5D | FF |  |  |
| 24 | AND #$F0 |  | 59 | F0 |  |  |
| 26 | OR #$AA |  | 5D | AA |  |  |
| 28 | XOR #$AA |  | 55 | AA |  |  |
| 2A | XOR #$AA |  | 55 | AA |  |  |
| 2C | CLRC |  | 40 |  |  |  |
| 2D | CSET |  | 48 |  |  |  |
| 2E | CLRC |  | 40 |  |  |  |
| 2F | CSET |  | 48 |  |  |  |
| 30 | LDA #$7F |  | 81 | 7F |  |  |
| 32 | ADDC #$01 |  | 49 | 01 |  |  |
| 34 | ADDC #$01 |  | 49 | 01 |  |  |
| 36 | ADDC #$01 | 28460 | 49 | 01 |  |  |
| 38 | SUBC #$01 |  | 71 | 01 |  |  |
| 39 | SUBC #$01 |  | 71 | 01 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |