On Exploring Algorithm Performance Between Von-Neumann and VLSI Custom-Logic Computing Architectures

Tiffany M. Mintz

James P. Davis, Ph.D.

South Carolina Alliance for Minority Participation

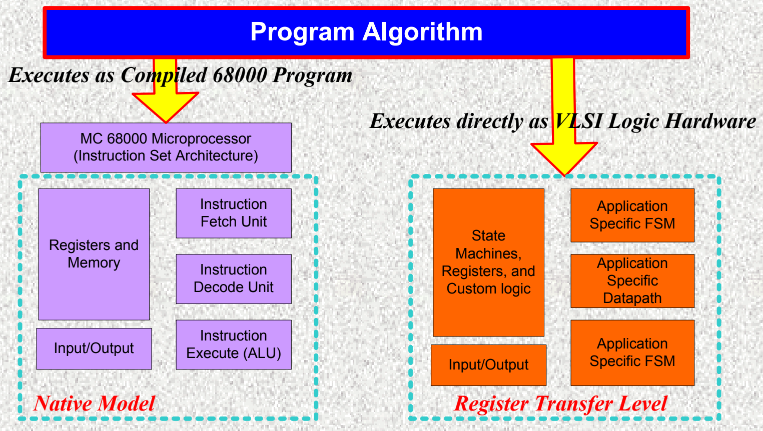
AMP

University of South Carolina

Statement of Research

- Explore the differences between microprocessor-based, and custom-VLSI logic-based, computing system models .
- Compare the difference in execution between microprocessor computing and custom logic computing architectures, using a set of benchmark algorithms.
- Write/select assembler programs that execute on a standard microprocessor (the Motorola 68000), and create corresponding custom logic architectures and designs for these same algorithms using an appropriate VLSI design method.
- Examine the differences in algorithmic processing between the two classes of computing architectures.
- Draw conclusions about the nature of algorithm processing between the two computing architecture models—the "old" and the "new".

Microprocessors vs. Custom Logic Computing Systems

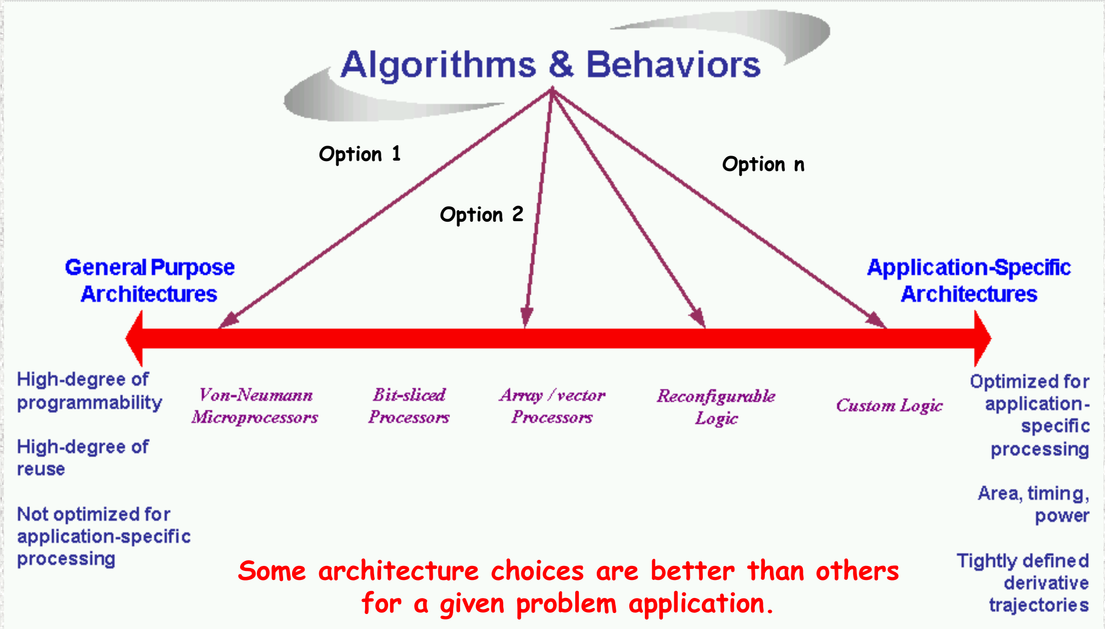


Program Algorithm
Executes as Compiled 68000 Program
Executes directly as VLSI Logic Hardware
MC 68000 Microprocessor (Instruction Set Architecture)
Registers and Memory
Instruction Fetch Unit
Instruction Decode Unit
Instruction Execute (ALU)
Input/Output
Native Model
State Machines, Registers, and Custom logic
Application Specific FSM
Application Specific Datapath
Application Specific FSM
Input/Output
Register Transfer Level

- A microprocessor is itself built from custom-designed VLSI logic, yet its programming model is based on the "fetch, decode, execute" paradigm pioneered by Dr. John Von Neumann almost 50 years ago. All standard microprocessor-based computer architectures are still built around this model.
- An application-specific custom VLSI computing system is not limited by the Von Neumann "bottleneck", as its control and data processing is inherently parallel, and its functions are distributed optimally across the VLSI device.

Algorithms and Architectures for Computing

Algorithms map onto different architectures over a continuum of design choices.

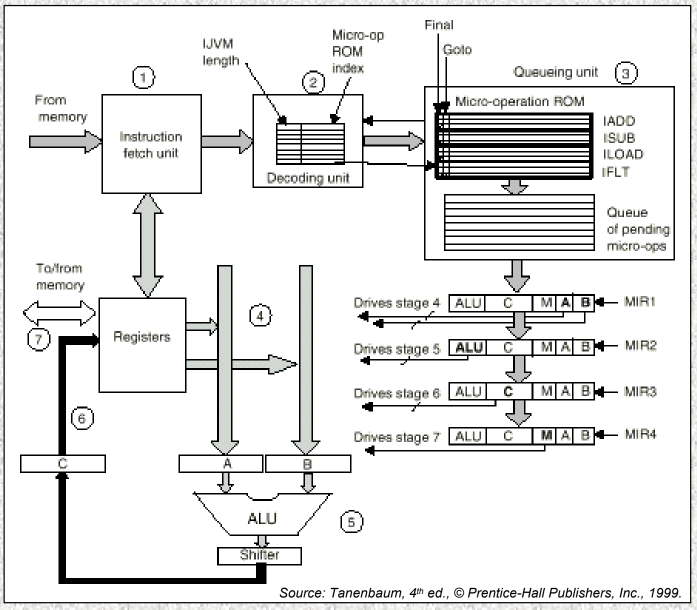


Algorithms & Behaviors
Option 1
Option 2
Option n
General Purpose Architectures
Application-Specific Architectures
High-degree of programmability
High-degree of reuse
Not optimized for application-specific processing
Von-Neumann Microprocessors
Bit-sliced Processors
Array / vector Processors
Reconfigurable Logic
Custom Logic
Optimized for application-specific processing
Area, timing, power
Tightly defined derivative trajectories

Some architecture choices are better than others for a given problem application.

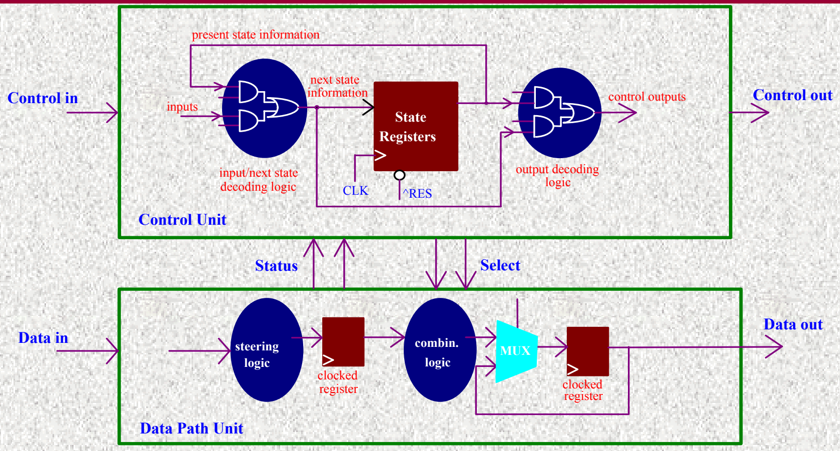
General Microprocessor Architecture

- A microprocessor architecture is based on the “fetch, decode, execute” cycle, that loops repeatedly.
- Applications exist as programs, loaded into memory along with required data.
- Program instructions are sequentially processed by the processor.
- The resources for all programs are shared, but dedicated to a specific executing program while running.



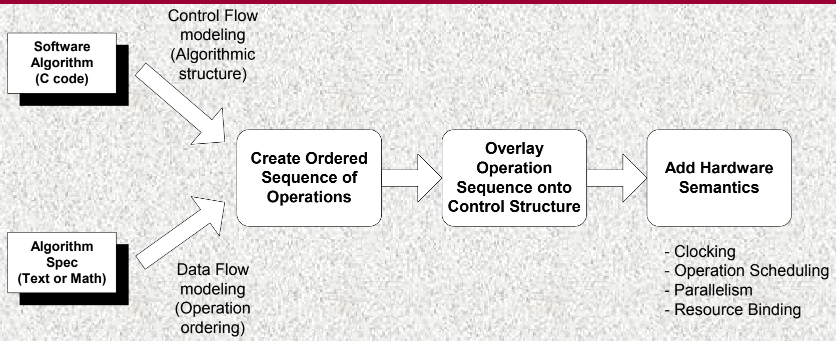
Source: Tanenbaum, 4th ed., © Prentice-Hall Publishers, Inc., 1999.

General Model of VLSI Architecture



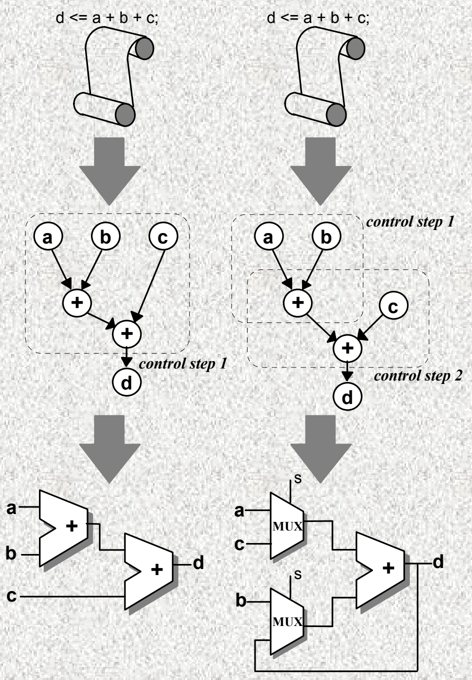
Control Units	Data Path Units
Modeled using Finite State Machine (FSM) model.	Modeled using Register-Transfer (RTL) model.
Defines clock-based sequencing of distributed actions in data path, or of those occurring external to the block.	Defines both synchronous and asynchronous transformations of data moving through the block. Data operations are distributed, with fine-grained parallelism.

Mapping Algorithms to VLSI Architecture


Software Algorithm (C code)
Control Flow modeling (Algorithmic structure)
Algorithm Spec (Text or Math)
Data Flow modeling (Operation ordering)
Create Ordered Sequence of Operations
Overlay Operation Sequence onto Control Structure
Add Hardware Semantics
- Clocking
- Operation Scheduling
- Parallelism
- Resource Binding

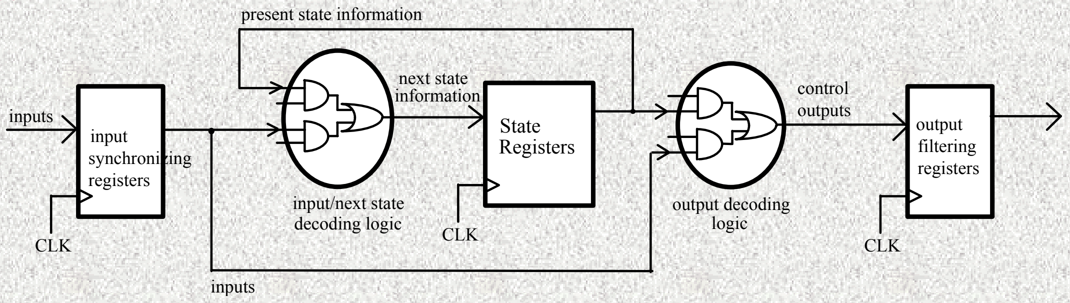
- Create Ordered Sequence of Operations.
  - Starting with Control Flow Graph (CFG) – If you are starting with the structure of an algorithm, such as from a block of C code, you can follow the structure of the algorithm as a basis for creating an ASM chart.
  - Starting with Data Flow Graph (CFG).
- Overlay Operation Sequence onto Control Structure.
- Add Hardware Semantics.
  - Quickly create a design model (correct by construction).
    - Create signal/bus declarations using Bus Table.
    - Draw the flow-chart description of the state machine.
    - Annotate states, conditions, cases, conditional output objects with RTN expressions (using assertions, assignments and macro-function assignments).
    - Define clocks, resets, and other synchronous/asynchronous event signaling.
  - Verify the Model (using digital cycle-based Simulator).

Exploring a VLSI Systems Architecture



- **Process starts with abstract description of algorithmic behavior written in C or some other language, with no timing info.**
  - Task #1: Compile source code into intermediate format, for example, control-flow graph, dataflow graph.
  - Task #2: schedule data operations to occur on specific control cycles, determined by clocking.
  - Task #3: allocate data operations to RTL components implied by use of language operators <+, -, *...>.
  - Task #4: bind specific operations to individual RTL components, to construct complete circuit topology.
- **We look for efficient architectures that speed up computation with minimal use of resources. This involves trading off speed versus resource usage.**

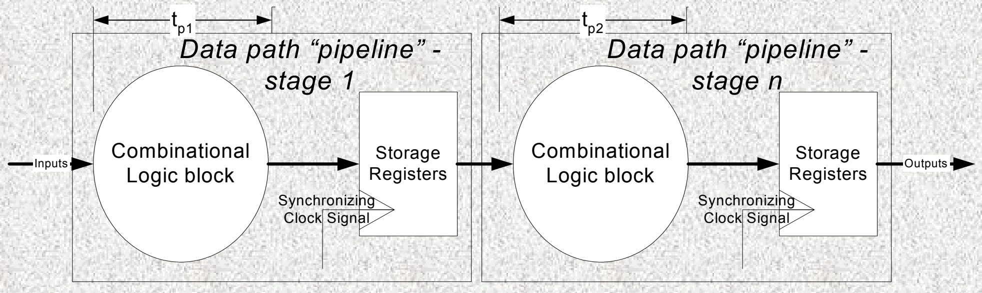
VLSI Systems Modeling-1


present state information
inputs
input synchronizing registers
CLK
next state information
input/next state decoding logic
State Registers
CLK
control outputs
output decoding logic
output filtering registers
CLK
inputs

Components of FSM Model

- State registers, input synchronization registers (optional) and output filter registers (optional).
- Next state decoding logic, and output decoding logic - combinational logic blocks.
- Input signals to the state machine, which are inputs to the next state and output decoding logic blocks (could be synchronized to clock with input registers).
- Next state information, which is generated as a result of input/next state decoding logic.
- Present state information, output from the state registers, which is fed back as an input to both next state and output decoding logic blocks.
- Outputs from the state machine - either generated synchronously from the output of the state registers (also used as present state information), or asynchronously as output of the output decoding logic block (which takes input and present state information to produce outputs). Could be filtered using output registers to eliminate possible signal transients.

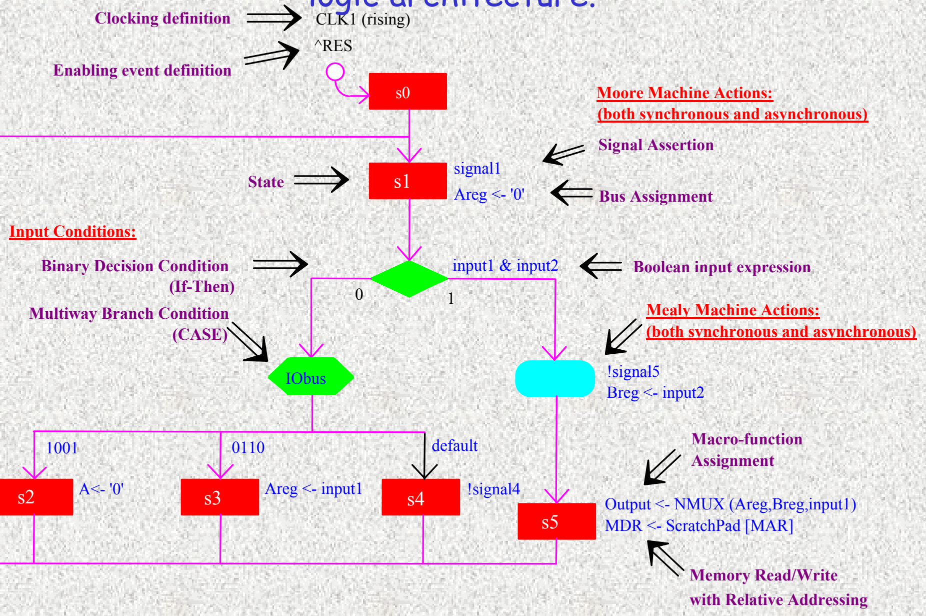
VLSI Systems Modeling-2


t_{p1}
Data path "pipeline" - stage 1
Inputs
Combinational Logic block
Storage Registers
Synchronizing Clock Signal
t_{p2}
Data path "pipeline" - stage n
Combinational Logic block
Storage Registers
Synchronizing Clock Signal
Outputs

- Use of memory elements in the data path to store signal values.
  - Purpose is to synchronize the behavior of complex circuits.
  - Benefits of circuit synchronization:
    - Eliminate unpredictability of output behavior due to timing skew.
    - Create signal stability, as they must have stable values for certain period of time.
    - Better isolate signals from noise transients.
- Use of memory to create complex control structures.
  - Controller sequences operations in the data path.
  - The sequencing is modeled as a finite state machine, represented as a graph structure.

The Algorithmic State Machine (ASM) Chart

Captures both the control path and data path design in a single design representation. It is used to model custom logic architecture.

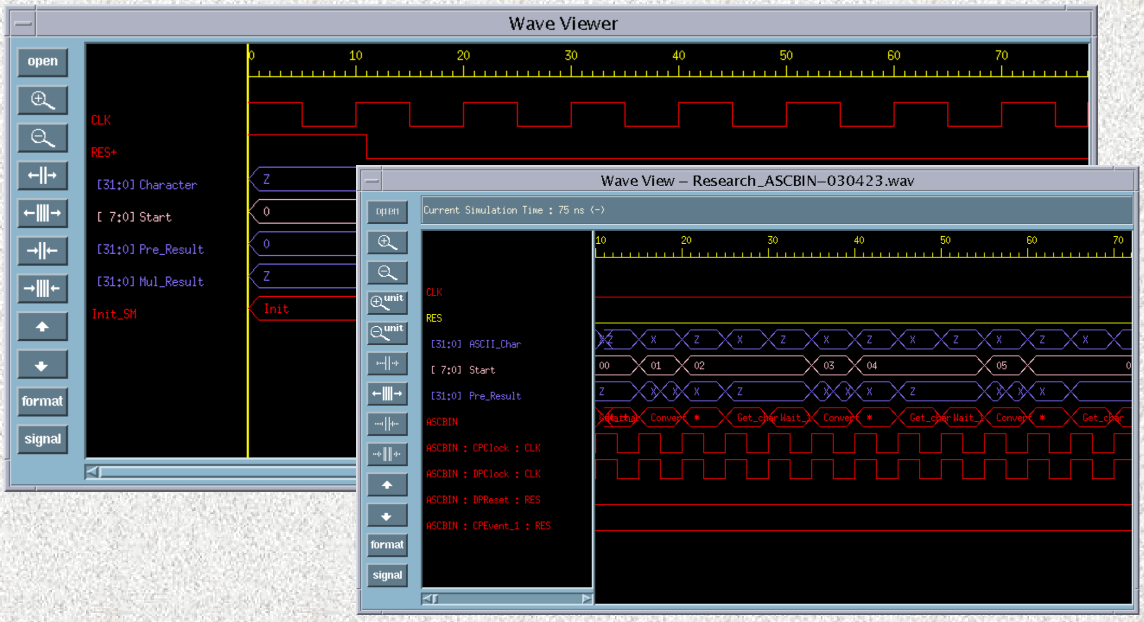


Benchmarking the Architecture Models

- **Using a 68000 microprocessor:**
  - A well-understood CPU model, as the micro is now 20 years old.
  - Used in CSCE 313 class for embedded systems design.
  - Select a set of baseline programs representing standard algorithms that have been studied in the past.
  - Using the cycle counts for each instruction, tally up the total cycles for the program, given the initial data elements defined for the benchmark programs (cf. MacKenzie, 1995).
- **Using the ASM design method:**
  - A well-understood custom logic design method, having been used for almost 30 years.
  - Used in CSCE 491, 611 classes for custom logic VLSI design.
  - Follow the same program algorithms, using RTL macro operations in place of 68000 instructions, yet inserting scheduling and clocking for synchronization.
  - Count the number of discrete states visited during the logic execution, given the same data elements defined for the baseline programs.

Counting Cycles in Custom Logic

Using the graphical view of the simulator waveform display, we can easily count the cycles required to execute an algorithm in a given VLSI architecture.



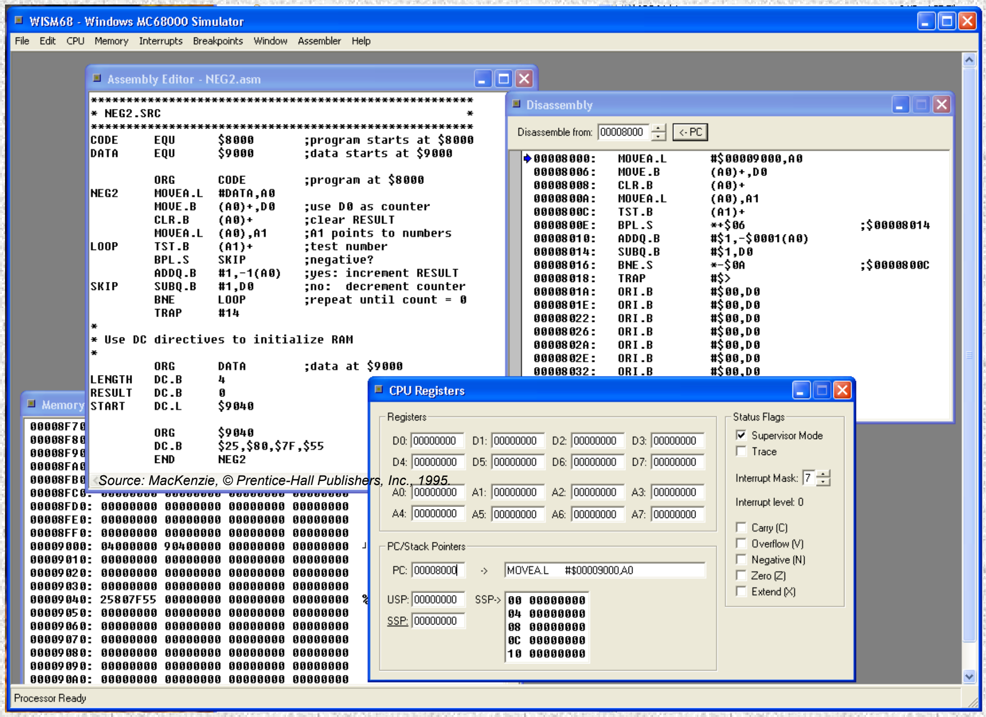
Benchmark Cycle Count Comparison

Benchmark	Clock Cycles CPU	Clock Cycles Custom Logic
NEG2: counting negative numbers in a sequence.	194	15
ASCBIN: converting ASCII string into equivalent binary number.	882	24
SORT: bubble sorting elements in a sequence.	782	35
SQRT: taking the square root of an unsigned integer.	1376	36

Benchmark Source: MacKenzie, © Prentice-Hall Publishers, Inc., 1995.

Note: lower is better!

The NEG2 Benchmark – 68K


WISM68 - Windows MC68000 Simulator
File Edit CPU Memory Interrupts Breakpoints Window Assembler Help
Assembly Editor - NEG2.asm
Disassembly
Memory
CPU Registers
Processor Ready

```
************************************************************
* NEG2.SRC                                                 *
************************************************************
CODE      EQU       $8000         ;program starts at $8000
DATA      EQU       $9000         ;data starts at $9000

          ORG       CODE          ;program at $8000
NEG2      MOVEA.L   #DATA,A0
          MOVE.B    (A0)+,D0      ;use D0 as counter
          CLR.B     (A0)+         ;clear RESULT
          MOVEA.L   (A0),A1       ;A1 points to numbers
LOOP      TST.B     (A1)+         ;test number
          BPL.S     SKIP          ;negative?
          ADDQ.B    #1,-1(A0)     ;yes: increment RESULT
SKIP      SUBQ.B    #1,D0         ;no:  decrement counter
          BNE       LOOP          ;repeat until count = 0
          TRAP      #14
*
* Use DC directives to initialize RAM
*
          ORG       DATA          ;data at $9000
LENGTH    DC.B      4
RESULT    DC.B      0
START     DC.L      $9040

          ORG       $9040
          DC.B      $25,$80,$7F,$55
          END       NEG2
```

Source: MacKenzie, © Prentice-Hall Publishers, Inc., 1995.

Disassemble from: 00008000 <- PC

```
00008000:   MOVEA.L    #$00009000,A0
00008006:   MOVE.B     (A0)+,D0
00008008:   CLR.B      (A0)+
0000800A:   MOVEA.L    (A0),A1
0000800C:   TST.B      (A1)+
0000800E:   BPL.S      *+$06                   ;$00008014
00008010:   ADDQ.B     #$1,-$0001(A0)
00008014:   SUBQ.B     #$1,D0
00008016:   BNE.S      *-$0A                   ;$0000800C
00008018:   TRAP       #$>
0000801A:   ORI.B      #$00,D0
0000801E:   ORI.B      #$00,D0
00008022:   ORI.B      #$00,D0
00008026:   ORI.B      #$00,D0
0000802A:   ORI.B      #$00,D0
0000802E:   ORI.B      #$00,D0
00008032:   ORI.B      #$00,D0
```

```
00008F70
00008F80
00008F90
00008FA0
00008FB0
00008FC0:
00008FD0: 00000000 00000000 00000000 00000000
00008FE0: 00000000 00000000 00000000 00000000
00008FF0: 00000000 00000000 00000000 00000000
00009000: 04000000 90400000 00000000 00000000
00009010: 00000000 00000000 00000000 00000000
00009020: 00000000 00000000 00000000 00000000
00009030: 00000000 00000000 00000000 00000000
00009040: 25807F55 00000000 00000000 00000000  %
00009050: 00000000 00000000 00000000 00000000
00009060: 00000000 00000000 00000000 00000000
00009070: 00000000 00000000 00000000 00000000
00009080: 00000000 00000000 00000000 00000000
00009090: 00000000 00000000 00000000 00000000
000090A0: 00000000 00000000 00000000 00000000
```

Registers

D0: 00000000 D1: 00000000 D2: 00000000 D3: 00000000
D4: 00000000 D5: 00000000 D6: 00000000 D7: 00000000
A0: 00000000 A1: 00000000 A2: 00000000 A3: 00000000
A4: 00000000 A5: 00000000 A6: 00000000 A7: 00000000

PC/Stack Pointers

PC: 00008000 -> MOVEA.L #$00009000,A0
USP: 00000000 SSP-> 00 00000000
SSP: 00000000 04 00000000
08 00000000
0C 00000000
10 00000000

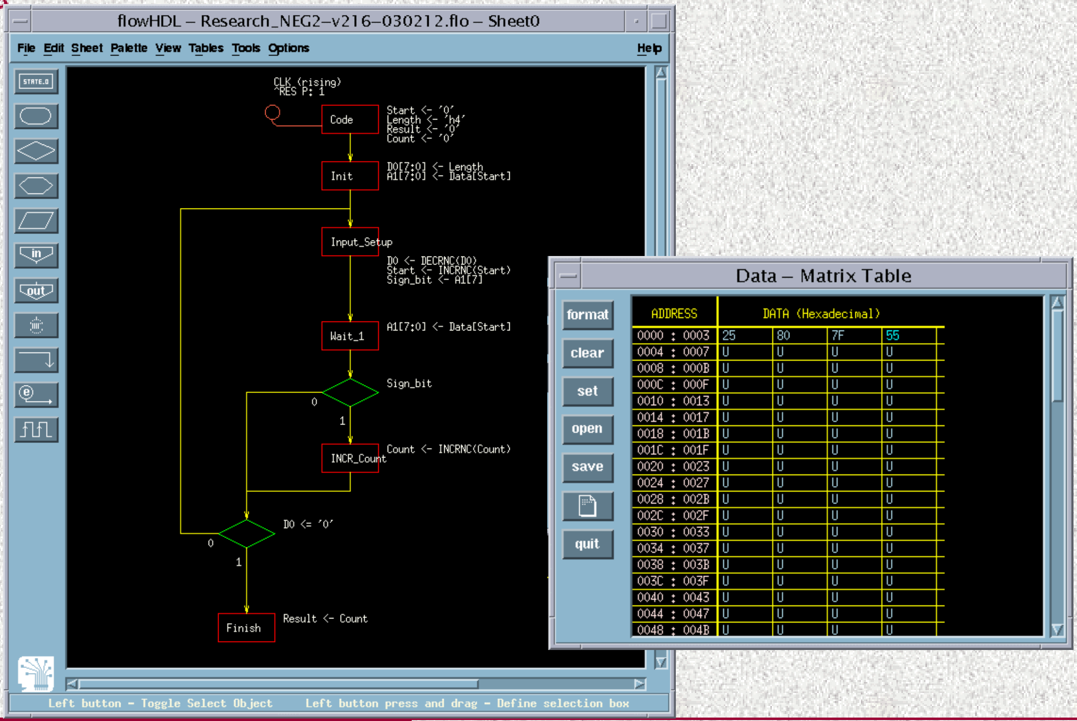
Status Flags

[✓ Supervisor Mode]
☐ Trace
Interrupt Mask: 7
Interrupt level: 0
☐ Carry (C)
☐ Overflow (V)
☐ Negative (N)
☐ Zero (Z)
☐ Extend (X)

VLSI Systems Design Lab
Computer Science and Engineering

UNIVERSITY OF SOUTH CAROLINA

The NEG2 Benchmark - ASM

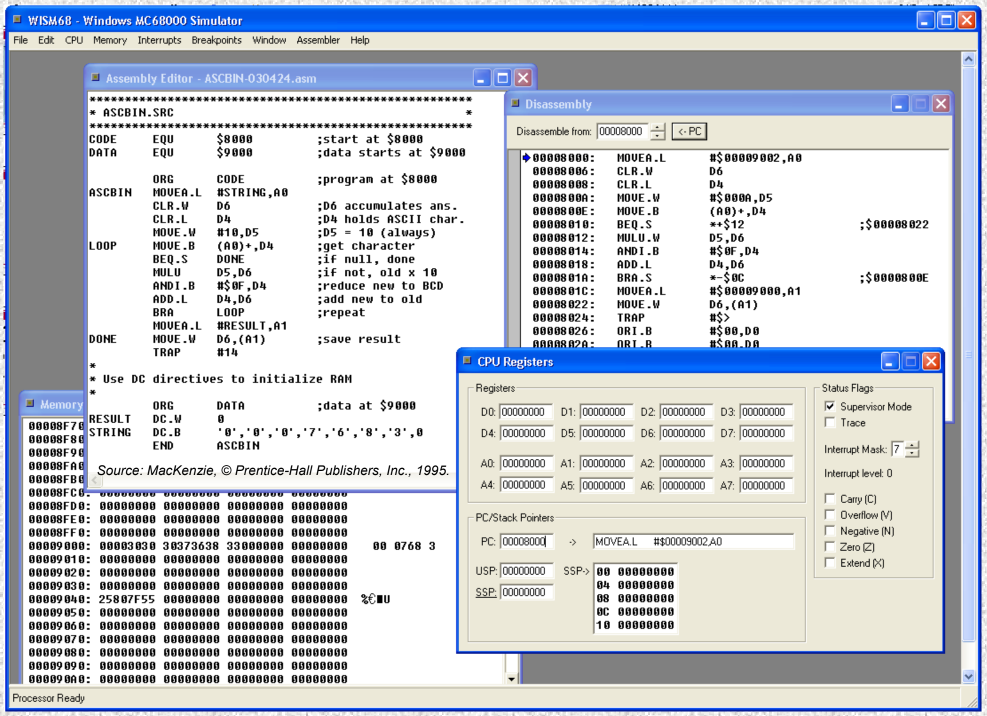


ADDRESS	DATA (Hexadecimal)			
0000 : 0003	25	80	7F	55
0004 : 0007	U	U	U	U
0008 : 000B	U	U	U	U
000C : 000F	U	U	U	U
0010 : 0013	U	U	U	U
0014 : 0017	U	U	U	U
0018 : 001B	U	U	U	U
001C : 001F	U	U	U	U
0020 : 0023	U	U	U	U
0024 : 0027	U	U	U	U
0028 : 002B	U	U	U	U
002C : 002F	U	U	U	U
0030 : 0033	U	U	U	U
0034 : 0037	U	U	U	U
0038 : 003B	U	U	U	U
003C : 003F	U	U	U	U
0040 : 0043	U	U	U	U
0044 : 0047	U	U	U	U
0048 : 004B	U	U	U	U

VLSI Systems Design Lab
Computer Science and Engineering

UNIVERSITY OF SOUTH CAROLINA

The ASCBIN Benchmark – 68K


WISM68 - Windows MC68000 Simulator
File Edit CPU Memory Interrupts Breakpoints Window Assembler Help
Assembly Editor - ASCBIN-030424.asm
Disassembly
Memory
CPU Registers
Processor Ready

```
*********************************************************
* ASCBIN.SRC                                            *
*********************************************************
CODE     EQU      $8000          ;start at $8000
DATA     EQU      $9000          ;data starts at $9000

         ORG      CODE           ;program at $8000
ASCBIN   MOVEA.L  #STRING,A0
         CLR.W    D6             ;D6 accumulates ans.
         CLR.L    D4             ;D4 holds ASCII char.
         MOVE.W   #10,D5         ;D5 = 10 (always)
LOOP     MOVE.B   (A0)+,D4       ;get character
         BEQ.S    DONE           ;if null, done
         MULU     D5,D6          ;if not, old x 10
         ANDI.B   #$0F,D4        ;reduce new to BCD
         ADD.L    D4,D6          ;add new to old
         BRA      LOOP           ;repeat
         MOVEA.L  #RESULT,A1
DONE     MOVE.W   D6,(A1)        ;save result
         TRAP     #14
*
* Use DC directives to initialize RAM
*
         ORG      DATA           ;data at $9000
RESULT   DC.W     0
STRING   DC.B     '0','0','0','7','6','8','3',0
         END      ASCBIN
```

Source: MacKenzie, © Prentice-Hall Publishers, Inc., 1995.

Disassemble from: 00008000 <- PC

```
00008000:   MOVEA.L    #$00009002,A0
00008006:   CLR.W      D6
00008008:   CLR.L      D4
0000800A:   MOVE.W     #$000A,D5
0000800E:   MOVE.B     (A0)+,D4
00008010:   BEQ.S      *+$12                  ;$00008022
00008012:   MULU.W     D5,D6
00008014:   ANDI.B     #$0F,D4
00008018:   ADD.L      D4,D6
0000801A:   BRA.S      *-$0C                  ;$0000800E
0000801C:   MOVEA.L    #$00009000,A1
00008022:   MOVE.W     D6,(A1)
00008024:   TRAP       #$>
00008026:   ORI.B      #$00,D0
0000802A:   ORI.B      #$00,D0
```

```
00008F70
00008F80
00008F90
00008FA0
00008FB0
00008FC0:
00008FD0: 00000000 00000000 00000000 00000000
00008FE0: 00000000 00000000 00000000 00000000
00008FF0: 00000000 00000000 00000000 00000000
00009000: 00003030 30373638 33000000 00000000   00 0768 3
00009010: 00000000 00000000 00000000 00000000
00009020: 00000000 00000000 00000000 00000000
00009030: 00000000 00000000 00000000 00000000
00009040: 25807F55 00000000 00000000 00000000  %€■U
00009050: 00000000 00000000 00000000 00000000
00009060: 00000000 00000000 00000000 00000000
00009070: 00000000 00000000 00000000 00000000
00009080: 00000000 00000000 00000000 00000000
00009090: 00000000 00000000 00000000 00000000
000090A0: 00000000 00000000 00000000 00000000
```

Registers

D0:	00000000	D1:	00000000	D2:	00000000	D3:	00000000
D4:	00000000	D5:	00000000	D6:	00000000	D7:	00000000
A0:	00000000	A1:	00000000	A2:	00000000	A3:	00000000
A4:	00000000	A5:	00000000	A6:	00000000	A7:	00000000

PC/Stack Pointers

PC: 00008000 -> MOVEA.L #$00009002,A0

USP: 00000000 SSP->

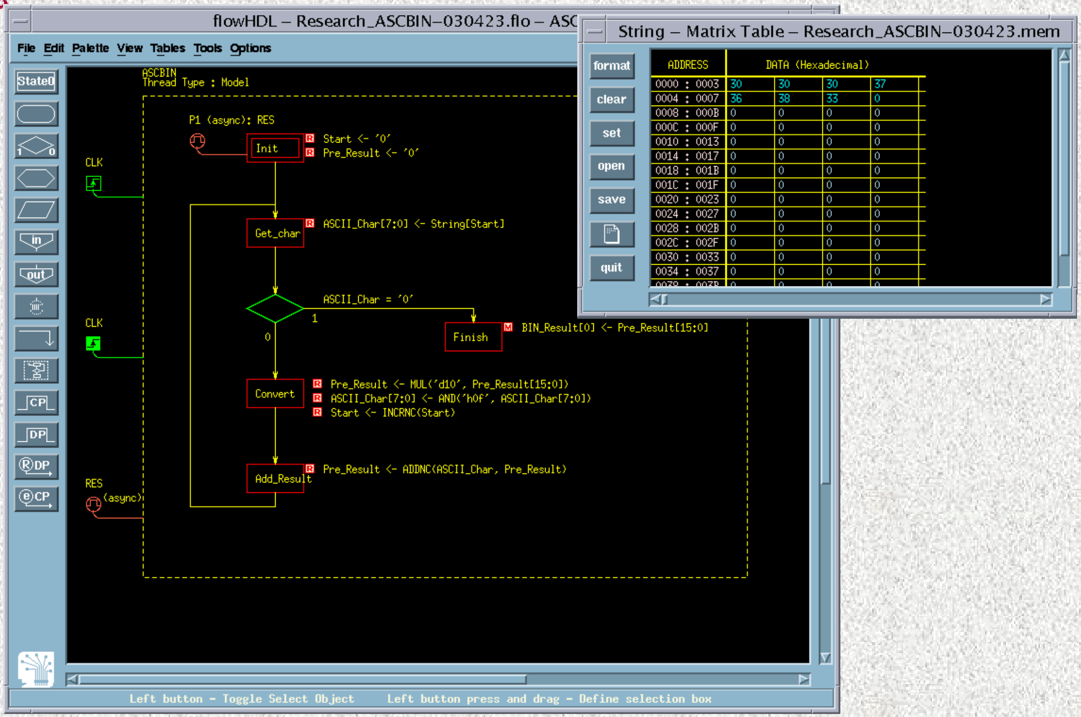
```
00  00000000
04  00000000
08  00000000
0C  00000000
10  00000000
```

SSP: 00000000

Status Flags

- ✓ Supervisor Mode
- ☐ Trace
- Interrupt Mask: 7
- Interrupt level: 0
- ☐ Carry (C)
- ☐ Overflow (V)
- ☐ Negative (N)
- ☐ Zero (Z)
- ☐ Extend (X)

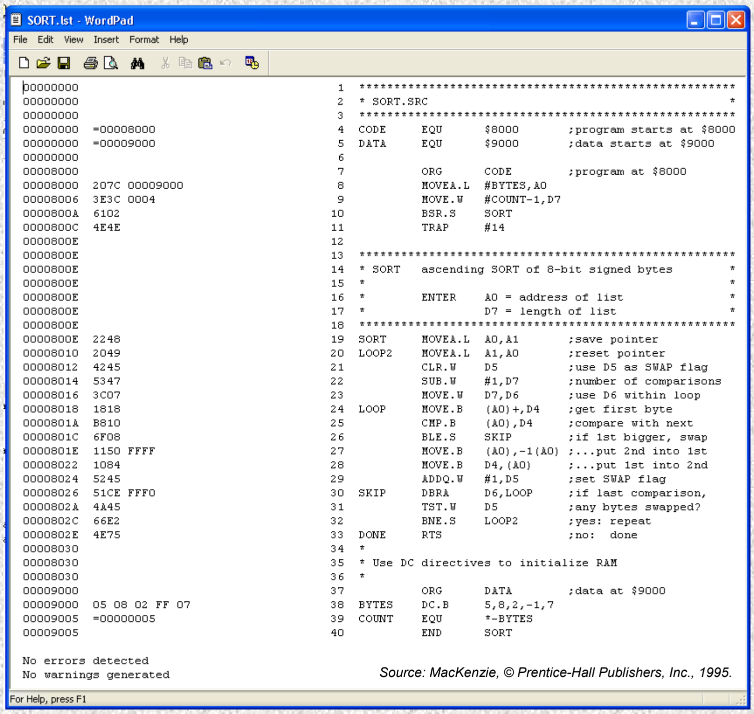
The ASCBIN Benchmark - ASM


flowHDL – Research_ASCBIN–030423.flo – ASC
File Edit Palette View Tables Tools Options
State0
CP
DP
ⓇDP
ⓔCP
ASCBIN
Thread Type : Model
P1 (async): RES
CLK
CLK
RES
(async)
Init
Get_char
Finish
Convert
Add_Result
1
0
Left button - Toggle Select Object   Left button press and drag - Define selection box
String – Matrix Table – Research_ASCBIN–030423.mem
format
clear
set
open
save
quit

```
Init:       Start <- '0'
            Pre_Result <- '0'
Get_char:   ASCII_Char[7:0] <- String[Start]
            ASCII_Char = '0'
Finish:     BIN_Result[0] <- Pre_Result[15:0]
Convert:    Pre_Result <- MUL('d10', Pre_Result[15:0])
            ASCII_Char[7:0] <- AND('h0f', ASCII_Char[7:0])
            Start <- INCRNC(Start)
Add_Result: Pre_Result <- ADDNC(ASCII_Char, Pre_Result)
```

ADDRESS	DATA (Hexadecimal)			
0000 : 0003	30	30	30	37
0004 : 0007	36	38	33	0
0008 : 000B	0	0	0	0
000C : 000F	0	0	0	0
0010 : 0013	0	0	0	0
0014 : 0017	0	0	0	0
0018 : 001B	0	0	0	0
001C : 001F	0	0	0	0
0020 : 0023	0	0	0	0
0024 : 0027	0	0	0	0
0028 : 002B	0	0	0	0
002C : 002F	0	0	0	0
0030 : 0033	0	0	0	0
0034 : 0037	0	0	0	0

The SORT Benchmark - 68K


SORT.lst - WordPad

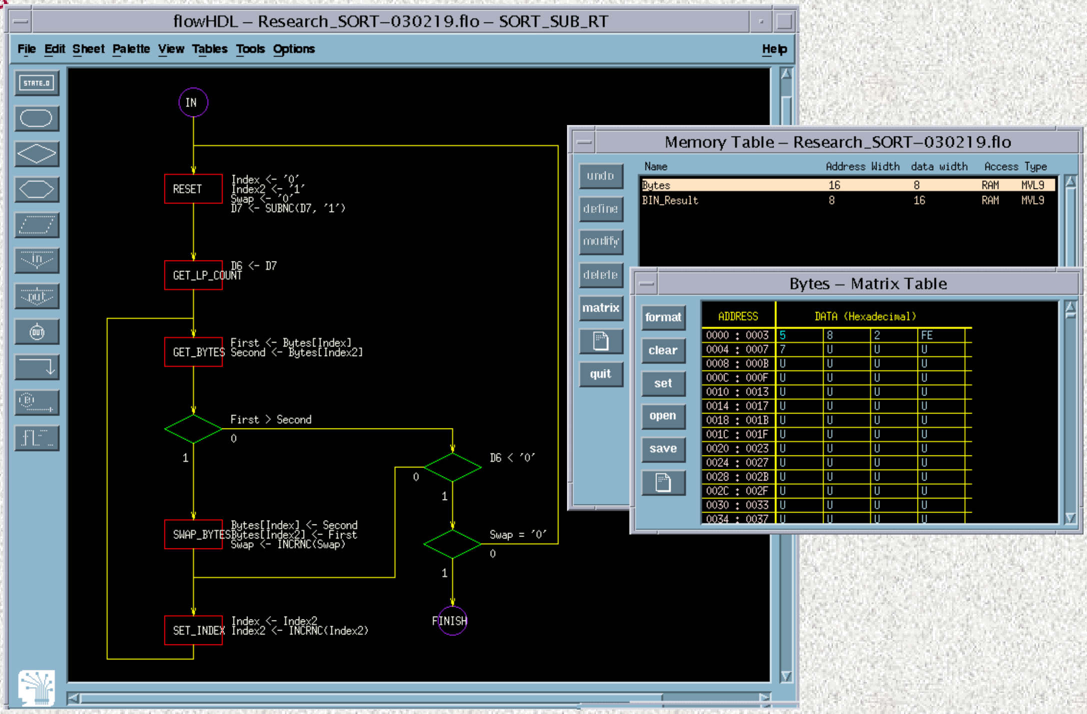
```
00000000                          ********************************************************
00000000                          * SORT.SRC                                             *
00000000                          ********************************************************
00000000  =00008000               CODE      EQU      $8000         ;program starts at $8000
00000000  =00009000               DATA      EQU      $9000         ;data starts at $9000
00000000
00008000                                    ORG      CODE          ;program at $8000
00008000  207C 00009000                     MOVEA.L  #BYTES,A0
00008006  3E3C 0004                         MOVE.W   #COUNT-1,D7
0000800A  6102                              BSR.S    SORT
0000800C  4E4E                              TRAP     #14
0000800E
0000800E                          ********************************************************
0000800E                          * SORT      ascending SORT of 8-bit signed bytes       *
0000800E                          *                                                      *
0000800E                          *         ENTER    A0 = address of list                *
0000800E                          *                  D7 = length of list                 *
0000800E                          ********************************************************
0000800E  2248                    SORT      MOVEA.L  A0,A1         ;save pointer
00008010  2049                    LOOP2     MOVEA.L  A1,A0         ;reset pointer
00008012  4245                              CLR.W    D5            ;use D5 as SWAP flag
00008014  5347                              SUB.W    #1,D7         ;number of comparisons
00008016  3C07                              MOVE.W   D7,D6         ;use D6 within loop
00008018  1818                    LOOP      MOVE.B   (A0)+,D4      ;get first byte
0000801A  B810                              CMP.B    (A0),D4       ;compare with next
0000801C  6F08                              BLE.S    SKIP          ;if 1st bigger, swap
0000801E  1150 FFFF                         MOVE.B   (A0),-1(A0)   ;...put 2nd into 1st
00008022  1084                              MOVE.B   D4,(A0)       ;...put 1st into 2nd
00008024  5245                              ADDQ.W   #1,D5         ;set SWAP flag
00008026  51CE FFF0               SKIP      DBRA     D6,LOOP       ;if last comparison,
0000802A  4A45                              TST.W    D5            ;any bytes swapped?
0000802C  66E2                              BNE.S    LOOP2         ;yes: repeat
0000802E  4E75                    DONE      RTS                    ;no:  done
00008030                          *
00008030                          * Use DC directives to initialize RAM
00008030                          *
00009000                                    ORG      DATA          ;data at $9000
00009000  05 08 02 FF 07          BYTES     DC.B     5,8,2,-1,7
00009005  =00000005               COUNT     EQU      *-BYTES
00009005                                    END      SORT

No errors detected
No warnings generated
```

Source: MacKenzie, © Prentice-Hall Publishers, Inc., 1995.

For Help, press F1

The SORT Benchmark - ASM



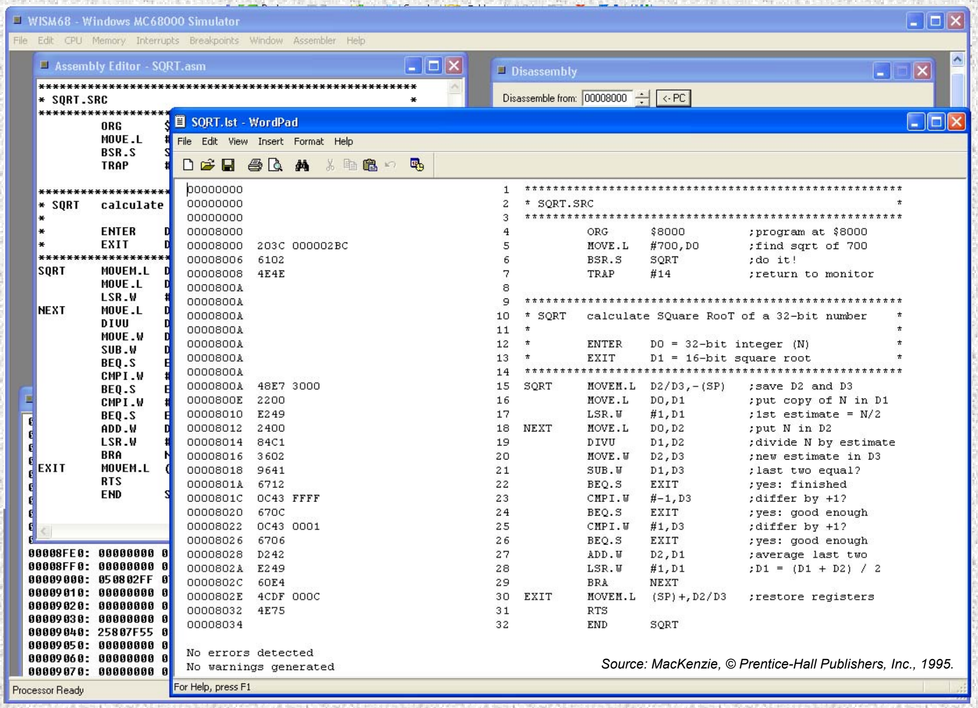
Memory Table – Research_SORT–030219.flo

Name	Address Width	data width	Access	Type
Bytes	16	8	RAM	MVL9
BIN_Result	8	16	RAM	MVL9

Bytes – Matrix Table

ADDRESS	DATA (Hexadecimal)			
0000 : 0003	5	8	2	FE
0004 : 0007	7	U	U	U
0008 : 000B	U	U	U	U
000C : 000F	U	U	U	U
0010 : 0013	U	U	U	U
0014 : 0017	U	U	U	U
0018 : 001B	U	U	U	U
001C : 001F	U	U	U	U
0020 : 0023	U	U	U	U
0024 : 0027	U	U	U	U
0028 : 002B	U	U	U	U
002C : 002F	U	U	U	U
0030 : 0033	U	U	U	U
0034 : 0037	U	U	U	U

The SQRT Benchmark – 68K



WISM68 - Windows MC68000 Simulator

SQRT.lst - WordPad

```
00000000                    ********************************************************
00000000                    * SQRT.SRC                                             *
00000000                    ********************************************************
00008000                             ORG      $8000        ;program at $8000
00008000  203C 000002BC              MOVE.L   #700,D0      ;find sqrt of 700
00008006  6102                       BSR.S    SQRT         ;do it!
00008008  4E4E                       TRAP     #14          ;return to monitor
0000800A
0000800A                    ********************************************************
0000800A                    * SQRT   calculate SQuare RooT of a 32-bit number      *
0000800A                    *                                                      *
0000800A                    *        ENTER    D0 = 32-bit integer (N)              *
0000800A                    *        EXIT     D1 = 16-bit square root              *
0000800A                    ********************************************************
0000800A  48E7 3000         SQRT     MOVEM.L  D2/D3,-(SP)  ;save D2 and D3
0000800E  2200                       MOVE.L   D0,D1        ;put copy of N in D1
00008010  E249                       LSR.W    #1,D1        ;1st estimate = N/2
00008012  2400              NEXT     MOVE.L   D0,D2        ;put N in D2
00008014  84C1                       DIVU     D1,D2        ;divide N by estimate
00008016  3602                       MOVE.W   D2,D3        ;new estimate in D3
00008018  9641                       SUB.W    D1,D3        ;last two equal?
0000801A  6712                       BEQ.S    EXIT         ;yes: finished
0000801C  0C43 FFFF                  CMPI.W   #-1,D3       ;differ by +1?
00008020  670C                       BEQ.S    EXIT         ;yes: good enough
00008022  0C43 0001                  CMPI.W   #1,D3        ;differ by +1?
00008026  6706                       BEQ.S    EXIT         ;yes: good enough
00008028  D242                       ADD.W    D2,D1        ;average last two
0000802A  E249                       LSR.W    #1,D1        ;D1 = (D1 + D2) / 2
0000802C  60E4                       BRA      NEXT
0000802E  4CDF 000C         EXIT     MOVEM.L  (SP)+,D2/D3  ;restore registers
00008032  4E75                       RTS
00008034                             END      SQRT

No errors detected
No warnings generated
```

```
00008FE0: 00000000 0
00008FF0: 00000000 0
00009000: 050802FF 0
00009010: 00000000 0
00009020: 00000000 0
00009030: 00000000 0
00009040: 25807F55 0
00009050: 00000000 0
00009060: 00000000 0
00009070: 00000000 0
```

Processor Ready

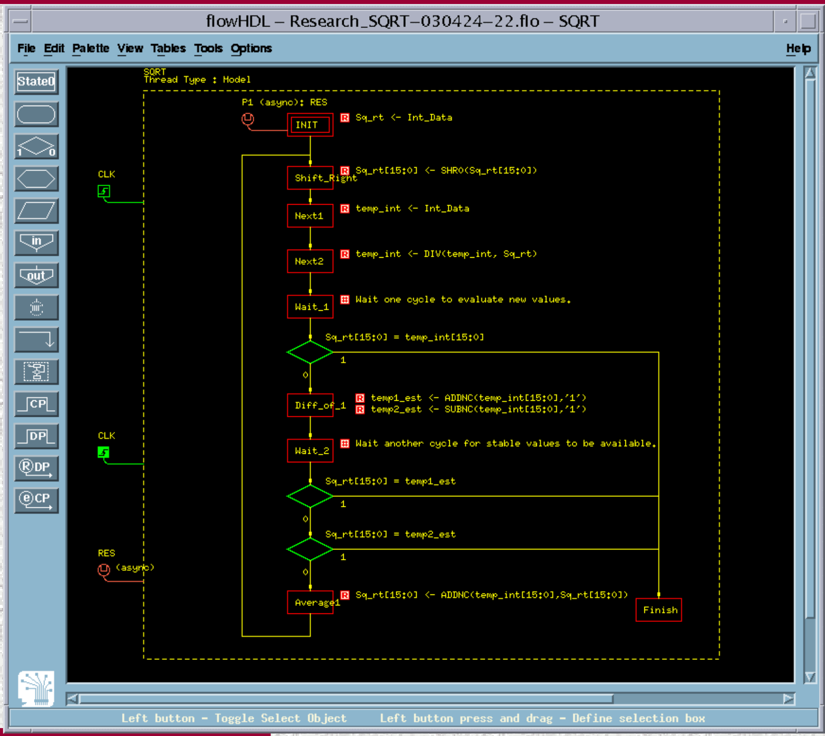
For Help, press F1

Source: MacKenzie, © Prentice-Hall Publishers, Inc., 1995.

VLSI Systems Design Lab
Computer Science and Engineering

UNIVERSITY OF SOUTH CAROLINA

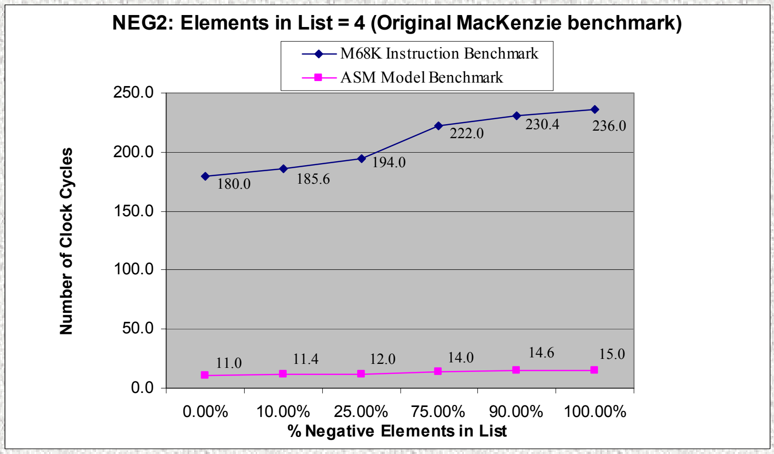
The SQRT Benchmark - ASM


flowHDL – Research_SQRT–030424–22.flo – SQRT
File Edit Palette View Tables Tools Options Help
SQRT
Thread Type : Model
P1 (async): RES
INIT
Sq_rt <- Int_Data
CLK
Shift_Right
Sq_rt[15:0] <- SHR0(Sq_rt[15:0])
Next1
temp_int <- Int_Data
Next2
temp_int <- DIV(temp_int, Sq_rt)
Wait_1
Wait one cycle to evaluate new values.
Sq_rt[15:0] = temp_int[15:0]
1
0
Diff_of_1
temp1_est <- ADDNC(temp_int[15:0],'1')
temp2_est <- SUBNC(temp_int[15:0],'1')
CLK
Wait_2
Wait another cycle for stable values to be available.
Sq_rt[15:0] = temp1_est
1
0
Sq_rt[15:0] = temp2_est
1
0
RES
(async)
Average1
Sq_rt[15:0] <- ADDNC(temp_int[15:0],Sq_rt[15:0])
Finish
Left button - Toggle Select Object
Left button press and drag - Define selection box

VLSI Systems Design Lab
Computer Science and Engineering

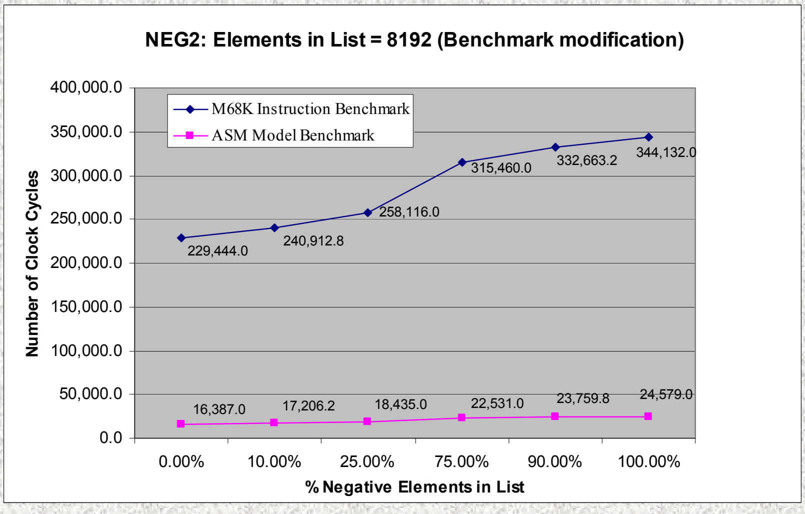
UNIVERSITY OF SOUTH CAROLINA

Comparing Cycle Counts – NEG2



Using the MacKenzie benchmark data set of N=4 elements, we look at two pieces of information: (1) what is the difference in the cycle counts between the different computing architecture styles; and, (2) what is the rate of change in cycle counts if we increased the number of negative elements in the sequence of length N that we needed to add to the running count.

Comparing Cycle Counts – NEG2



Extending the original benchmark scope with N=8k elements, we look at the two questions again: (1) what is the cycle count difference between the microprocessor and custom logic executions; and, (2) what rate of change in cycle counts occurs as we increase the number of negative elements in the sequence as a percentage of the total elements.

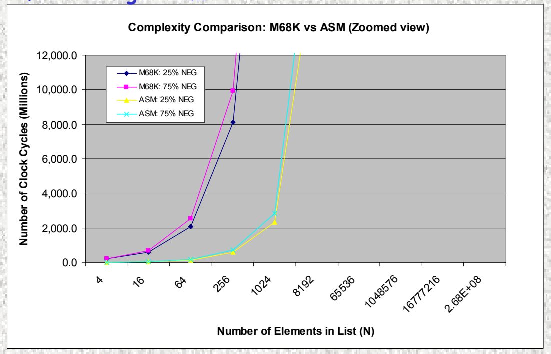
Comparing Complexity - NEG2

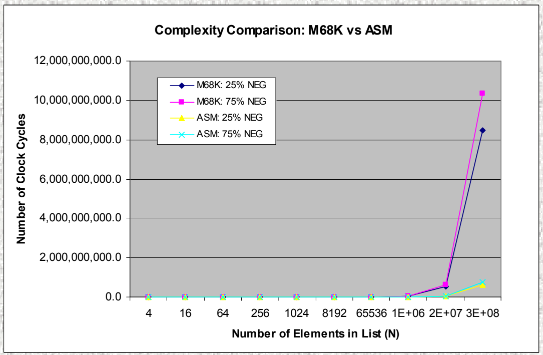
		Number of Elements Being Processed (N)									
Benchmark	Reference	Size1	Size2	Size3	Size4	Size5	Size6	Size7	Size8	Size9	Size10
NEG2:	Fig E, p. 135	*factor=2**2*	*factor=2**4*	*factor=2**6*	*factor=2**8*	*factor=2**10*	*factor=2**13*	*factor=2**16*	*factor=2**20*	*factor=2**24*	*factor=2**28*
Count Negative Numbers	N =	4	16	64	256	1024	8192	65536	1048576	16777216	268435456
in a List	%Neg Elements(NE) =	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	%Neg Elements(NE) =	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%	10.00%
	%Neg Elements(NE) =	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%	25.00%
	%Neg Elements(NE) =	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%	75.00%
	%Neg Elements(NE) =	90.00%	90.00%	90.00%	90.00%	90.00%	90.00%	90.00%	90.00%	90.00%	90.00%
	%Neg Elements(NE) =	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
M68K Instruction Benchmark											
Negative Numbers in List:	Tc = 68 + 18*N + 10*(1-NE) + 24*NE cycles										
Tc=	0.00%	180.0	516.0	1,860.0	7,236.0	28,740.0	229,444.0	1,835,076.0	29,360,196.0	469,762,116.0	7,516,192,836.0
	10.00%	185.6	538.4	1,949.6	7,594.4	30,173.6	240,912.8	1,926,826.4	30,828,202.4	493,250,218.4	7,892,002,474.4
	25.00%	194.0	572.0	2,084.0	8,132.0	32,324.0	258,116.0	2,064,452.0	33,030,212.0	528,482,372.0	8,455,716,932.0
	75.00%	222.0	684.0	2,532.0	9,924.0	39,492.0	315,460.0	2,523,204.0	40,370,244.0	645,922,884.0	10,334,765,124.0
	90.00%	230.4	717.6	2,666.4	10,461.6	41,642.4	332,663.2	2,660,829.6	42,572,253.6	681,155,037.6	10,898,479,581.6
	100.00%	236.0	740.0	2,756.0	10,820.0	43,076.0	344,132.0	2,752,580.0	44,040,260.0	704,643,140.0	11,274,289,220.0
ASM Model Benchmark											
Negative Numbers in List:	Tc = 2 + N(2 + NE(1)) +1 cycles										
Tc=	0.00%	11.0	35.0	131.0	515.0	2,051.0	16,387.0	131,075.0	2,097,155.0	33,554,435.0	536,870,915.0
	10.00%	11.4	36.6	137.4	540.6	2,153.4	17,206.2	137,628.6	2,202,012.6	35,232,156.6	563,714,460.6
	25.00%	12.0	39.0	147.0	579.0	2,307.0	18,435.0	147,459.0	2,359,299.0	37,748,739.0	603,979,779.0
	75.00%	14.0	47.0	179.0	707.0	2,819.0	22,531.0	180,227.0	2,883,587.0	46,137,347.0	738,197,507.0
	90.00%	14.6	49.4	188.6	745.4	2,972.6	23,759.8	190,057.4	3,040,873.4	48,653,929.4	778,462,825.4
	100.00%	15.0	51.0	195.0	771.0	3,075.0	24,579.0	196,611.0	3,145,731.0	50,331,651.0	805,306,371.0

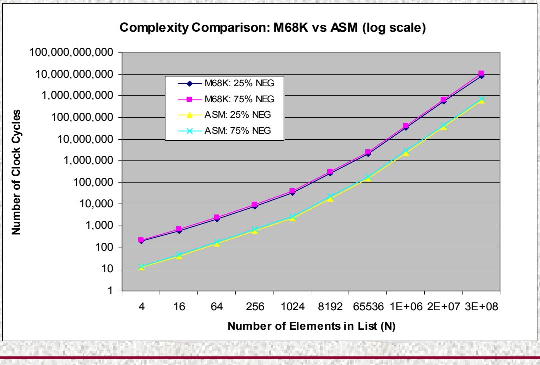
Extending the original benchmark scope yet again by varying N, we look at the two questions: (1) what is the cycle count difference between the microprocessor and custom logic executions as N increases; and, (2) what rate of change in cycle counts occur as we increase the number of negative elements in the sequence as a percentage of the total elements while N grows? Does *% Neg Elements* matter?

Comparing Complexity- NEG2

What we learned: (1) the custom logic architecture is an order of magnitude more efficient, in terms of number of clock cycles, in performing the same computational task, (2) this ratio is consistent as N grows large, but (3) the "number of probes" of the list (represented by *% of negatives* for NEG2) does not seem to be a relevant metric of complexity for this algorithm.







What we have yet to calculate: (1) the actual time to perform the task, given some clock frequencies for the microprocessor and the custom logic device, (2) the computational throughput (calculations per unit time), which can be affected by pipelining of data operations, CPU instruction caching, etc.

Future Work

- Extend the scope of coverage to incorporate time complexity analysis of the other benchmarks, to see what happens to computation with both architecture models as N grows large, and as we increase the number of “probe points” in the data set at each value of N.
- Examine the time complexity characteristics O(n), Ω(n) and other identified metrics for VLSI custom logic architectures in other benchmarks that have different algorithmic control structures.
- Modify the custom logic models by exploiting inherent parallelism afforded by VLSI device structure. Here, we might exploit parallelism & pipelining to increase performance of the VLSI design, by changing the “shape” of the algorithm.
- Explore more generally how time complexity and other characteristics are affected by different architecture topologies for various standard algorithms in both models.

References

- I. S. MacKenzie, *The 68000 Microprocessor*, Prentice-Hall Publishers, Inc., 1995.
- A. Tanenbaum, Structured Computer Organization, 4th ed., Prentice-Hall Publishers, Inc., 1999.
- *flowHDL Reference Manual*, Knowledge Based Silicon Corporation, 1996.
- Buell, D. A., Davis, J. P., and G. Quan, "Reconfigurable Computing Applied to Problems in Communications Security", in *Proceedings MAPLD-2002 Military Applications of Programmable Logic Devices*, Advanced Physics Laboratory, Johns Hopkins University, 2002.
- Davis, J., Nagarkar, S., and J. Mathewes, “High-level Design of On-Chip Systems for Integrated Control and Data Path Applications”, *Proceedings Design SuperCon 1996 On-chip System Design Conference*, Hewlett Packard Company & Integrated Systems Design Magazine, 1996.
- D. Wood, *Data Structures, Algorithms, and Performance*, Addison Wesley Publishing Co., Inc., 1993.
- B. Codenotti and M. Leoncini, *Introduction to Parallel Processing*, Addison-Wesley Publishing Co., Inc., 1993.
- M. A. Weiss, *Data Structures and Algorithm Analysis in* C, Benjamin/Cummings Publishing Co., Inc., 1993.
- L. Banachowski, A. Kreczmar and W. Rytter, *Analysis of Algorithms and Data Structures*, Addison-Wesley Publishing Co., Inc., 1991.