NASA Technical Memorandum

NASA TM-86517

HARDWARE MATH FOR THE 6502 MICROPROCESSOR

By Ralph Kissel and James Currie

Information and Electronic Systems Laboratory
Science and Engineering Directorate

July 1985

National Aeronautics and
Space Administration

George C. Marshall Space Flight Center

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
NASA TM-86517		

4. TITLE AND SUBTITLE	5. REPORT DATE
Hardware Math for the 6502 Microprocessor	July 1985
	6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT #
Ralph Kissel and James Currie	

9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. WORK UNIT NO.
George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812	11. CONTRACT OR GRANT NO.
	13. TYPE OF REPORT & PERIOD COVERED
12. SPONSORING AGENCY NAME AND ADDRESS	Technical Memorandum
National Aeronautics and Space Administration Washington, D.C. 20546	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES

Prepared by Information and Electronic Systems Laboratory, Science and Engineering Directorate.

16. ABSTRACT

A floating-point arithmetic unit is described which is being used in the Ground Facility for Large Space Structures Control Verification (GF/LSSCV). The experiment uses two complete inertial measurement units and a set of three gimbal torquers in a closed loop to control the structural vibrations in a flexible test article (beam). A 6502 (8-bit) microprocessor controls four AMD 9511A floating-point arithmetic units to do all the computation in 20 milliseconds.

17. KEY WORDS	18. DISTRIBUTION STATEMENT
6502 Microprocessor Hardware Math Floating-point Coprocessor AIM 65	Unclassified – Unlimited

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	18	NTIS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

TECHNICAL MEMORANDUM

HARDWARE MATH FOR THE 6502 MICROPROCESSOR

INTRODUCTION

Floating-point arithmetic is generally a time-consuming task, especially on an 8-bit microprocessor. The system described here is the result of a growing arithmetic workload in a real-time control system using the given 6502 microprocessor. Four AMD 9511A's (Intel 8231A) were used in parallel and connected directly to the 6502. The 6502 (in the Rockwell AIM 65) was clocked at 2 MHz and the 9511's at 4 MHz.

Originally, the AIM 65 was to read data from six gyros and six accelerometers (two complete inertial navigation systems) and from two resolvers, then send these data to a central computer once each 20 ms. Then, more computation became necessary as strapdown algorithms, control algorithms, and finally, everything except mass storage was added to the software. The system described here does all this arithmetic in the required time and can be adapted to any 6502 system to give an arithmetic speedup of about 100 times over BASIC.

HARDWARE

Figure 1 shows the timing diagrams for a 6502 at 2 MHz and a 9511 at 4 MHz. The 9511 is asynchronous which makes interfacing easier. The biggest hardware problem with the 6502 is that it cannot be stopped during a write. The problem with timing is to obtain the 25 ns minimum between WR going high and CS, A0 going high.

Figure 2 shows the one-9511 interface for a typical 6502 system and Figure 3 shows the full four-9511 interface as described in this report.

Reference 1 gives details on interfacing a 9511 to a 6502 by using a 6522 VIA interface chip. This method is straightforward to implement, but has so much overhead that it is primarily useful only for trigonometric functions.

Reference 2 gives details of another method for directly connecting the 9511 to a 6502 (OSI Superboard II). The method was optimized to the OSI board, but could probably have been adapted as needed had it been discovered soon enough.

DMA is generally the fastest method that could be used, but was not worked out for the 6502. It would be quite complex and may not actually be any faster than the method used here.

The actual hardware connections between the 6502 bus and the 9511's went through a Computerist, Inc., DRAM board which was already part of this AIM 65 system. Chip select (CS) was also done by the DRAM board, although this was only for convenience. Another, more direct method to generate the control signals could be used so long as the same timing is kept.

Interfaces for other microprocessors (using one 9511) are generally simpler than that for the 6502 and are already published by AMD.

SOFTWARE

In the four-APU system all software was handwritten. There is about 5K object code and 3K BASIC. Reference 2 used a compiler, but none was written here. More than one APU implies overlap and critical timing considerations to get maximum speed and efficiency, so a compiler did not seem feasible.

A simulation was written for the desktop HP 9845A, which plotted timelines for the CPU and each of the four APU's. Busy and idle times are all clearly shown, so code could be inserted if needed. Any APU overlap (sending to a busy APU, etc.) was also flagged (see Appendix).

Another simulation of the actual arithmetic was written for the HP 9845A so numerical correctness could be checked. This simulation did the same calculations, in the same order, as the AIM 65 system itself. Still another simulation (done by a contractor as part of an overall system study) was made to do the original theoretical algorithms. Eventually, all results agreed and the timeline indicated a worst-case time less than 20 ms.

Hardware is arranged so that each APU, or any combination of APU's, has a unique address. Data are sent by doing a LDA data then STA address which takes 4 μs total. One floating-point number for an APU contains 4 bytes so the minimum time to load is 16 μs. A command will usually only require 3 μs to send – LDA immediate, STA address. A data read requires 4 μs.

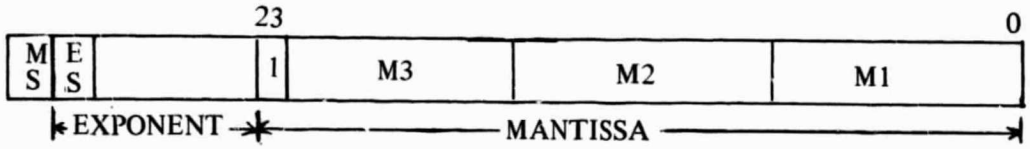
Two methods were used to determine when an APU is finished. One is to check the busy bit in the status register of a particular APU. The other is reading a special address which contains the four APU END lines as the lower four bits and zeros in the upper four. This extra hardware was added to increase CPU response time and reduce software volume since more than one APU status check would normally require a separate read and check for each overlapped APU. Waiting for all end lines to go low requires only one read of this added register.

Sometimes no check of APU status was made at all, since it was known that the APU could not be busy. Interrupts were not used since response time is much too slow (at least 21 μs at 2 MHz), plus I/O uses interrupts also (conflict).

The hardware was designed such that the CPU will halt when an APU read is attempted and that APU is busy. It will remain stopped until the APU is no longer busy. On a write, however, no attempt is made to stop the CPU since it won't stop during a write. It will stop only on the next non-write instruction. This causes a continual problem and requires implementing a check to be sure the APU is not busy before writing to it. Figure 4 is an example of how an APU can be used.

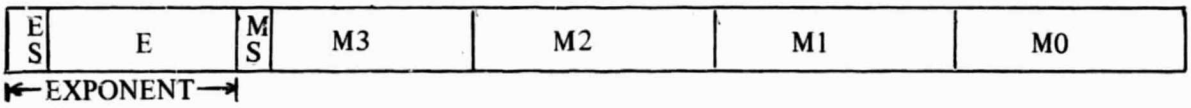
There is another 65C2 peculiarity that has to be kept in mind when the memory map is being laid out. It only applies when indexed instructions are used across a page boundary and reflects the 6502 design rules. During one microcycle the address of the incorrect preceding page is actually put on the bus and could activate whatever was there. This is always a read, but an APU could still be triggered if it happened to have that address.

The APU's have one floating-point format – AIM 65 BASIC memory has another. Since BASIC was used as the controlling language, conversion in both directions had to be made. All constants and one-time calculations were done in BASIC and converted to APU format (Fig. 5). When results from an APU are displayed by BASIC they must be converted to BASIC format (Fig. 6). The APU format is:



Exponent is unbiased 2's complement 7 bits (–64 to +63)
Bit 23 ≡ 1 except for zero which is all zeros.
Mantissa sign: 0 = +, 1 = –

and the BASIC format is



Exponent: $81 = +1; $80 = 0; 7F = –1 etc. (biased).
Bit 7 of M3 is Mantissa sign: 0 = +, 1 = –.
0 ≡ all five bytes = 0.

Instruction sequences were always done to make maximum use of otherwise idle CPU time and even APU time when one is free. Registers can be loaded and other calculations and operations inserted to eliminate the idle time. The timeline simulation was used to do this efficiently.

Minimum software and maximum speed required a tradeoff of in-line versus subroutine code. In-line is considerably faster (16 μs versus 22 μs + overhead), but subroutines require much less memory (EPROM eventually), so subroutines were generally used since enough time was thought to be available (it just was). There were also many tradeoffs on types of assembly instructions to use to keep the number of subroutines to a minimum, but still allow fast execution. Absolute indexed mode was generally used, since some versatility is available without the speed penalty of indirect instructions. For a particular subroutine, the assembler can increment the absolute part with the index then not needing changed at run time.

Error checks were not made in this application. It seemed that too much time would be wasted looking for the only two errors that could occur – overflow and underflow. Underflow should not be ignored, actually, it must be worked around. Underflow does not result in zero, but instead, a change in exponent sign. This could be disastrous. So, scaling must be done to prevent it, it must be checked, or safety checks must be made in case it occurs. Underflow occurs at about $\pm 2.7 \times 10^{-20}$, overflow about $\pm 9.2 \times 10^{18}$, and overflow means there's a hardware problem which will immediately appear elsewhere in the results.

Software reset of the APU's was not implemented. The AIM 65 has this as a manual button, if it should ever be required. If it is required, it means there is a noise problem or a hardware problem that must be fixed.

PERFORMANCE

The quad system in this application is running at 0.026 MFLOP. This includes CPU overhead, APU idle time, APU stack manipulations,etc. An 8086-8087 at 5 MHz would do about 0.021 MFLOP. A 6 MHz 68000-16081 combination should be around 0.050 MFLOP and an HP9000 using compiled BASIC without floating-point hardware about 0.075 MFLOP.

Overhead generally doubles the time it takes to do any particular operation. Data must be moved into and out of the APU, stack changes must be made to obtain higher efficiency, and the CPU must sometimes wait for an APU to finish.

More APU's could be tied to one APU (2 MHz), but four is about the limit for simple arithmetic. If trigonometric functions were the primary requirement, then up to 10 APU's could be kept busy.

FORTH would be much better than BASIC for most applications where the maximum speed is not required. Pure assembly code squeezes an extra factor of 5 to 10 out of the hardware. With FORTH, little or no assembly code would be needed.

Using all in-line code, (MACROS) could increase the overall speed by maybe 10 percent, but at a cost of three times the original program memory (5K to 15K here).

SUMMARY

The quad-9511 system described here works rather well, but is time consuming to program and debug. A typical application would only have one APU, which would be much easier to use. One APU would also make writing a compiler a feasible and useful task.

Overlapped APU operation makes program changes something to be done with great care. Also, the carry flag and the X-register sometimes are expected to retain their value through several subroutines. The CPU is often doing non-APU operations while some APU's are still busy. Overflow and underflow may be a problem and must be considered, particularly in a real-time system. The LSS unit continues to operate as expected.

REFERENCES

1. DeJong, Marvin L.: Interfacing the AM9511 Arithmetic Processing Unit. COMPUTE!, November-December 1980.

2. Hart, John E.: Microcrunch: An Ultra-fast Arithmetic Computing System. MICRO, August 1981 (Part 1), September 1981 (Part 2).

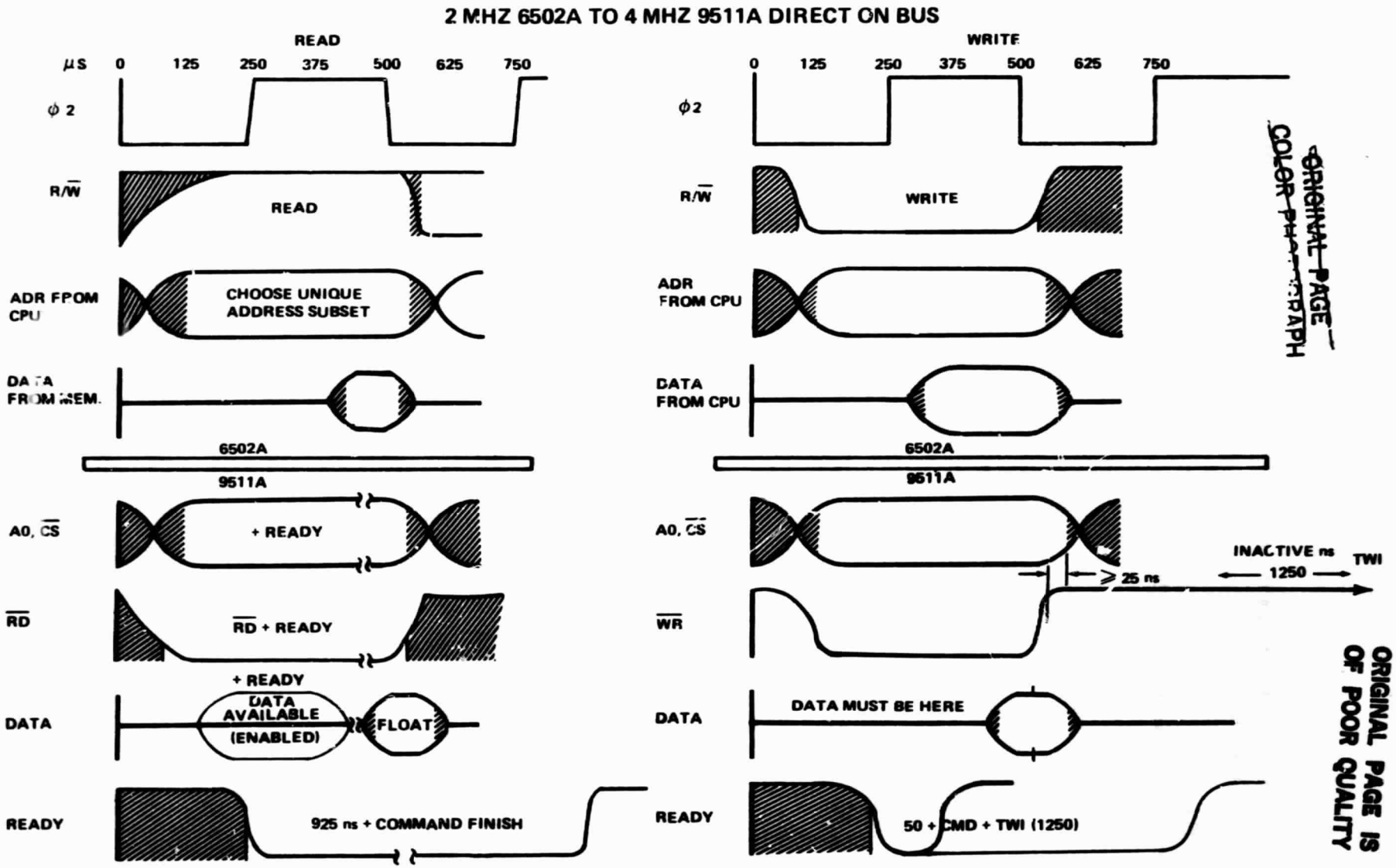
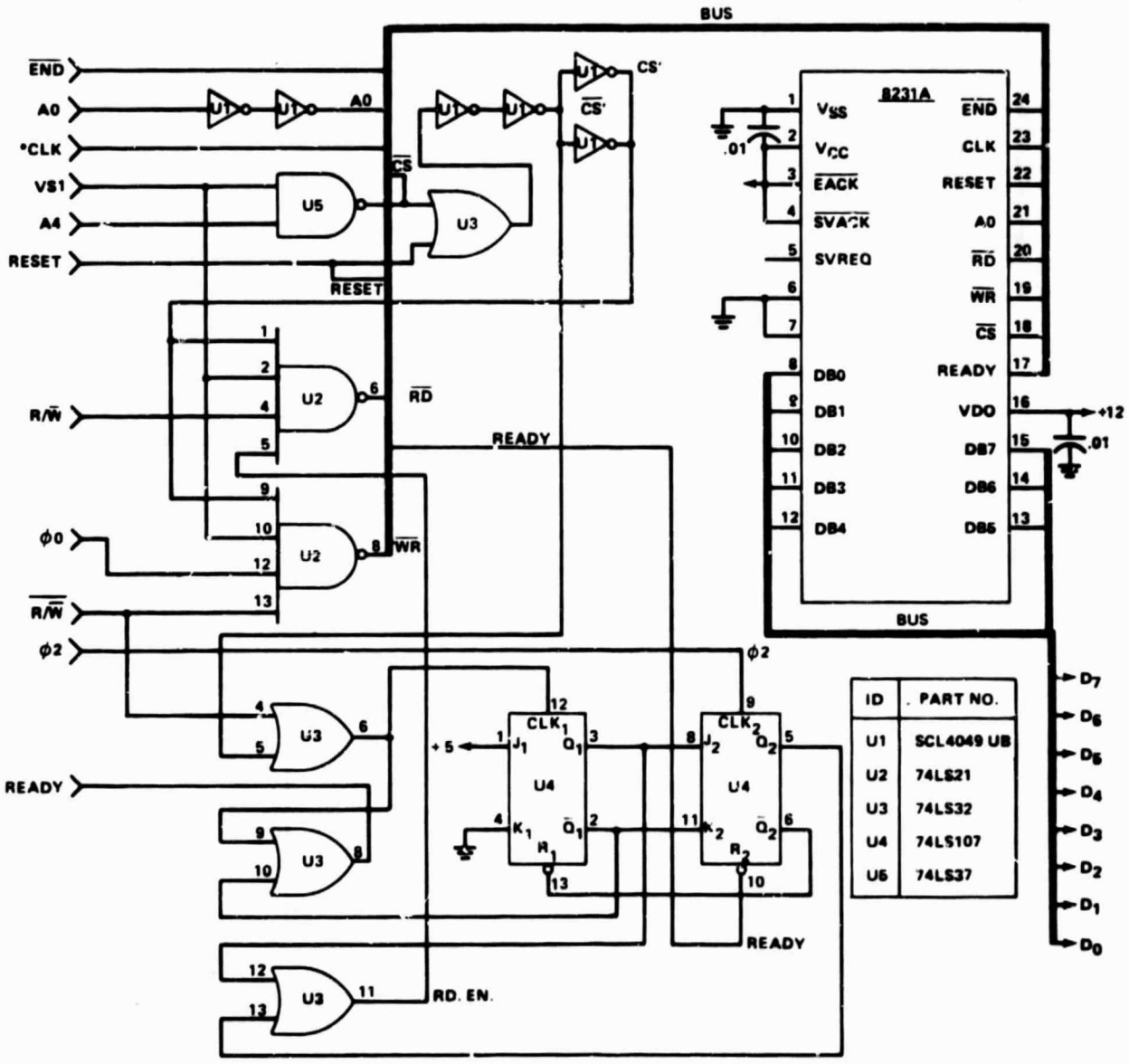


Figure 1. Timing diagrams.

RULES

1) CHIP ENABLE $\overline{CS}$ AND A0 MUST BE STABLE BEFORE $\overline{RD}$ OR $\overline{WR}$ GO LOW.
2) $\overline{RD}$ MUST RETURN HIGH $\geq$ 0 NS BEFORE $\overline{CS}$ GOES HIGH OR A0 CHANGES.
3) $\overline{WR}$ MUST RETURN HIGH AT LEAST 25 NS BEFORE $\overline{CS}$ GOES HIGH OR A0 CHANGES.
4) ON READ CYCLE NOT READY MUST BE SET LO BEFORE ϕ 2 GOES HIGH.



ID	PART NO.
U1	SCL4049 UB
U2	74LS21
U3	74LS32
U4	74LS107
U5	74LS37

*CLK THE CLOCK FREQUENCY FOR THE 8231A IS 4 MHZ.

ϕ0 THE AIM 65 CHIP CLOCK IS 2 MHZ. OR LESS.

ADDRESS LINES (INCLUDES VS1) SHOULD BE BUFFERED.

Figure 2. Single-unit interface.

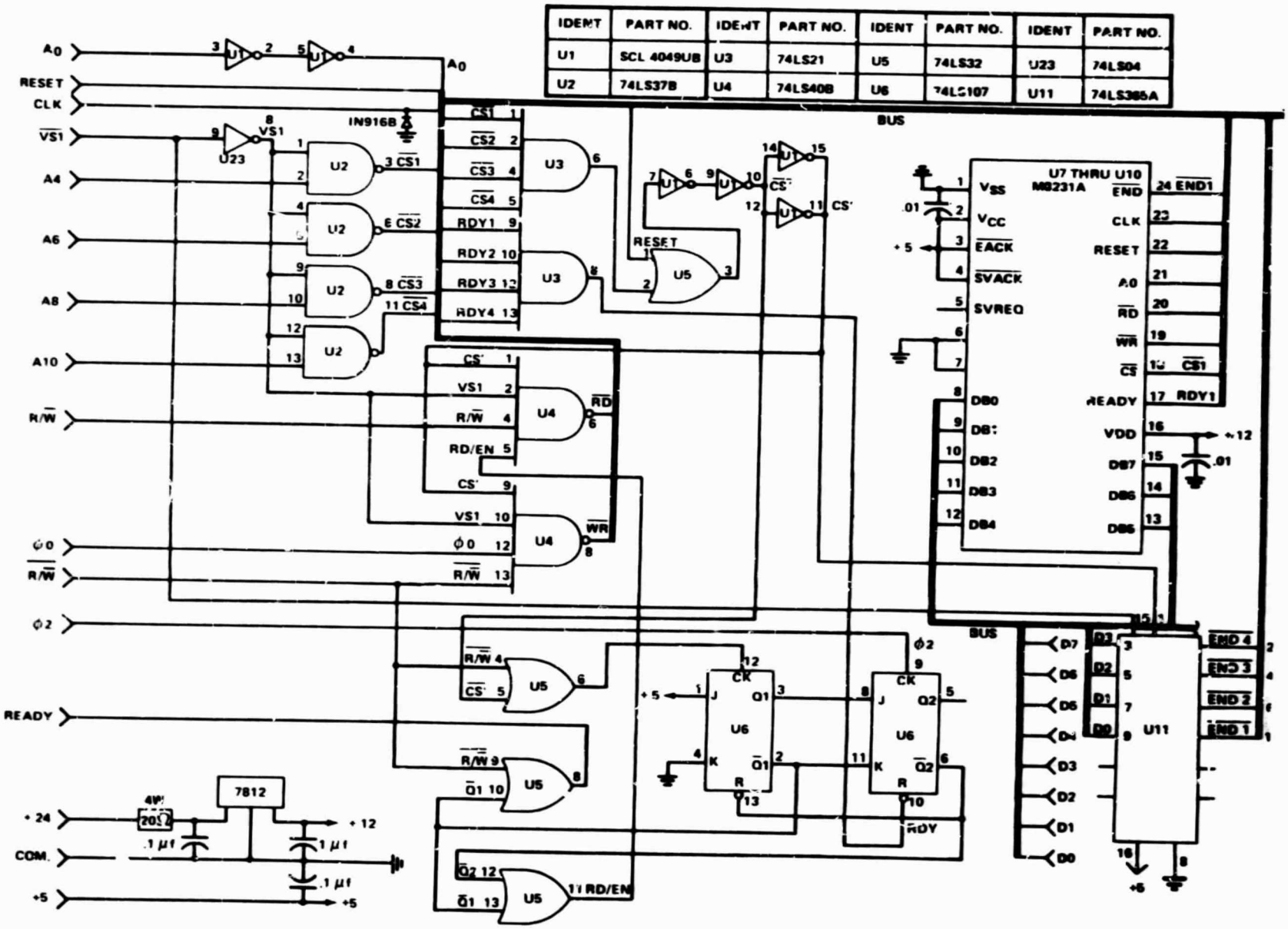


Figure 3. Quad-unit interface.

```
    NUM1=74
    NUM2=61
    APUD=$9010    (DATA)
    APUC=$9011    (CMD)
    DATA0=$700    (RESULT)
    *=$800
    LDA #NUM1
    STA APUD
    LDA #0
    STA APUD
    STA APUD
    STA APUD      (FIRST #)
    LDA #$1C
    STA APUC      (FLOAT)
A1  BIT APUC
    BMI A1        (WAIT)
    LDA #NUM2
    STA APUD
    LDA #0
    STA APUD
    STA APUD
    STA APUD      (SECOND #)
    LDA #$1C
    STA APUC      (FLOAT)
A2  BIT APUC
    BMI A2        (WAIT)
    LDA #$12
    STA APUC      (FMUL)
A3  BIT APUC
    BMI A3        (WAIT)
    LDA #$1E
    STA APUC      (FIXED)
A4  BIT APUC
    BMI A4        (WAIT)
    LDA APUD
    STA DATA0
    LDA APUD
    STA DATA0+1
    LDA APUD
    STA DATA0+2
    LDA APUD
    STA DATA0+3   (RESULT)
    BRK
    .END
```

Figure 4. Single-unit multiply example.

```
      *=$3000          USES 42 USEC
      BADR=143         BASIC VAR LOC (5 BYTES)
      BUF82=$FE        8231 BUFFER (4 BYTES)
T082  LDY #0
      LDA (BADR),Y
      BNE T01
      LDY #3
      LDA #0           CHECK IF EXP =0
 T03  STA (BUF82),Y    AND IF SO SET
      DEY              ALL 8231=0
      BNE T03
      RTS
 T01  CMP #$C0         (C0 --> NO OVF
      BCC T04          CLEAR --> (
      LDA #$3F         OVF
      BNE T05
 T04  CMP #$40         >=40 --> NO UF
      BCS T06          SET --> >=
      LDA #$40         UF
 T05  ASL A            LEFT ONE SO CAN ROR
      TAX              NEED ACC
      INY              Y=1
      LDA (BADR),Y
      ASL A            PUT M. SIGN INTO CARRY
      TXA              GET OVF OR UF
      ROR A            APPEND CARRY TO IT
      DEY              Y=0
      STA (BUF82),Y
      LDA #$FF
      LDY #3           ALL ONES IN
 T02  STA (BUF82),Y    MANTISSA
      DEY              (INCL BIT 23)
      BNE T02
      RTS
 T06  ASL A            LINE UP FOR LATER ROR
      TAX              NEED ACC SAVED
      INY              Y=1
      LDA (BADR),Y
      CMP #$80         SET CARRY IF BIT 7 SET
      ORA #$80         BIT 7 TO 1
      STA (BUF82),Y
      TXA              RECALL DATA
      ROR A            APPEND CARRY
      DEY              Y=0
      STA (BUF82),Y
      LDY #2
      LDA (BADR),Y     DIRECT TRANSFER OF
      STA (BUF82),Y    THESE BYTES
      INY              (TRUNCATE 4 TO 3)
      LDA (BADR),Y
      STA (BUF82),Y
      RTS
```

Figure 5. AIM-to-8231 format conversion.

```
      *=$3100          USES 54 USEC
TOBA  LDY #4
      LDA #0
      STA (BADR),Y     ALWAYS ZERO
      DEY              Y=3
 BA1  LDA (BUF82),Y
      STA (BADR),Y
      DEY              3,2,1 BUT NOT Y=0
      BNE BA1
      AND #$FF         CHECK IF=0 ALWAYS (SET FLAGS)
      BNE BA2
      STA (BADR),Y     BYTE ZERO ALSO=0
      RTS
 BA2  LDA (BUF82),Y    NOT=0
      BMI BA3          SKIP IF M.S. OK
      INY              Y=1
      LDA (BUF82),Y
      AND #$7F         M.S.=0
      STA (BADR),Y
      DEY              Y=0
 BA3  LDA (BUF82),Y
      AND #$7F         CLEAR M.S. BIT (EXP NOW)
      CMP #$40         SET CARRY IF >= $40
      BCS BA4
      ORA #$80         SET E.S.
 BA4  STA (BADR),Y
      RTS
```

Figure 6. 8231-to-AIM format conversion.

APPENDIX

TIMELINE SIMULATION

The entire simulation is somewhat lengthy, so only the subroutine that actually calculates the time intervals will be discussed. It can then be used as needed in large systems. All other subroutines eventually call this one. It calculates and plots CPU time plus the four APU times, idle as well as busy. It checks for illegal overlap, namely, writing to an already busy APU. Preventing such overlap is the software designer's responsibility since the hardware cannot prevent it.

There are six subroutine parameters. T1 is the total CPU time including any subroutine call overhead. The assumed normal situation is that a subroutine is being simulated. If not, then T1 is made negative and the call overhead time is not included. The user is expected to include the subroutine return overhead time whenever it occurs since this is CPU time. Subroutine call overhead is automatically added unless T1 is negative. The return overhead is not automatically handled because it often is separated from the first part of its subroutine and because it is easier to handle than the call overhead.

Dt is the delta time (CPU) until an APU is first operated. The subroutine call overhead, if any, is included in Dt. Both T1 and Dt are measured from the time before a subroutine call was initiated, if any.

The P value is the time that an APU is committed to a task, either executing an instruction, or loading or unloading a command or data. A minus sign means the CPU will wait until that APU is ready.

If Grf = 1, then the full graph will be drawn. If Grf = 0, only execution times will be available to be printed (by external routines). The graph shows busy time for the CPU and each of the four APU's. Available CPU and APU times are immediately visible. The final time is always available and it can be worst-case (typically) or otherwise, depending on needs. Other items such as percent utilization, efficiency, etc., of various schemes could be added.

Any sequence of machine code can be simulated with this subroutine. Pure CPU time can be done as well as all overlapped APU's. Routines may have to be split in various ways to make them fit. Variable times will need to be fixed, with, usually, either the minimum or maximum value.

Figure A-1 is the subroutine listing as written in HP 9845 BASIC. It assumes a 2 MHz 6502 and 4 MHz APU's. The calculation portion can easily be adapted to another machine, but the graphics may be more involved.

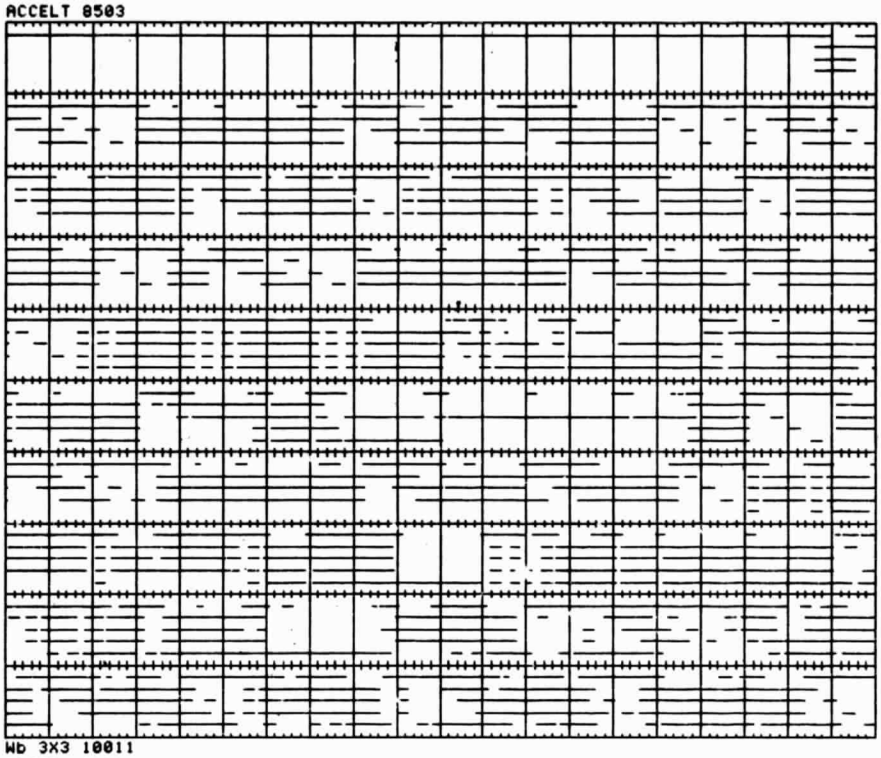
Figure A-2 shows the results when this subroutine is applied to the LSS project. Changes are easy to make and their effects easy to see. With so much overlap taking place, making a change often produces an unexpected result. Times used here are in microseconds and there are no fractional values permitted. Calculated final time is 19.5 ms worst case, and measured times were in the 18 ms range.

Earlier results indicated over 20 ms (22.1) and various otherwise idle times were put to use in achieving the reduction. Further reduction would be quite difficult. Only a 10 percent reduction as obtained here is probably not typical, since the original code was already tightly written.

```
1090 SUB C(T1,Dt,P1,P2,P3,P4)
1100 COM X1,X2,Y1,Y2,Z1,Z2,E1,E2,V1,C,Xx,Yy,Fadd,Fmul,Nop,Fsub,Flts,Ptof,Popf,Xc
hf,Chsf,Fixs,Grf
1110 Cc=C+3*(T1>=0)
1120 Dc=Dt-3*(T1>=0)
1130 M=MAX((X2-Cc)*(P1<0),(Y2-Cc)*(P2<0),(Z2-Cc)*(P3<0),(E2-Cc)*(P4<0),0)
1140 M=INT(INT((M+1)/3.5)*3.5)
1150 Mm=Cc+Dc+M
1160 Cx=Mm
1170 IF (P1>0) AND (X2>=Cx) THEN PRINT "APU1 OVERLAP";C;T1;Dt;X2-Mm
1180 IF (P2>0) AND (Y2>=Cx) THEN PRINT "APU2 OVERLAP";C;T1;Dt;Y2-Mm
1190 IF (P3>0) AND (Z2>=Cx) THEN PRINT "APU3 OVERLAP";C;T1;Dt;Z2-Mm
1200 IF (P4>0) AND (E2>=Cx) THEN PRINT "APU4 OVERLAP";C;T1;Dt;E2-Mm
1210 T1=ABS(T1)+M
1220 IF (P1=0) OR (P1=-1) THEN 1250
1230 IF X2<=C THEN X1=Mm
1240 X2=Mm+ABS(P1)
1250 IF (P2=0) OR (P2=-1) THEN 1280
1260 IF Y2<=C THEN Y1=Mm
1270 Y2=Mm+ABS(P2)
1280 IF (P3=0) OR (P3=-1) THEN 1310
1290 IF Z2<=C THEN Z1=Mm
1300 Z2=Mm+ABS(P3)
1310 IF (P4=0) OR (P4=-1) THEN 1340
1320 IF E2<=C THEN E1=Mm
1330 E2=Mm+ABS(P4)
1340 REM PRINT C;T1;Dt;M;Mm;P1;P2;P3;P4,X1;X2;Y1;Y2;Z1;Z2;E1;E2
1350 IF Grf=0 THEN C=C+T1
1360 IF Grf=0 THEN 1640
1370 Ty=Yy*Xx/6
1380 FOR T=C TO C+T1-1
1390 IF (T<>INT(T/Ty)*Ty) OR (T=0) THEN 1440
1400 DUMP GRAPHICS
1410 V1=-5
1420 GCLEAR
1430 GRID 10,6,0,Yy,5,1,2
1440 Tt=T MOD Xx
1450 Ts=Tt+1
1460 IF (Tt=0) AND (T<>0) THEN V1=V1+6
1470 IF (T>=Cc) AND (T<Cc+M) THEN 1500
1480 MOVE Tt,V1
1490 DRAW Ts,V1
1500 IF (T<X1) OR (T>=X2) THEN 1530
1510 MOVE Tt,V1+1
1520 DRAW Ts,V1+1
1530 IF (T<Y1) OR (T>=Y2) THEN 1560
1540 MOVE Tt,V1+2
1550 DRAW Ts,V1+2
1560 IF (T<Z1) OR (T>=Z2) THEN 1590
1570 MOVE Tt,V1+3
1580 DRAW Ts,V1+3
1590 IF (T<E1) OR (T>=E2) THEN 1620
1600 MOVE Tt,V1+4
1610 DRAW Ts,V1+4
1620 NEXT T
1630 C=T
1640 SUBEXIT
```

Figure A-1. Timeline subroutine.

AB 3X3 919
ACCELB 1816
WT 3X3 3175
QTGET 4072
QGET 4417
AT 3X3 5945
TROT 6842
ACCELT 8503



WB 3X3 10011
QBGET 10908
QGET 11253
ZANDWB 12781
ZANDWT 14508
TORK 16258
END 1st 6 17341
END 2nd 6 18684
FINAL TIME 19455

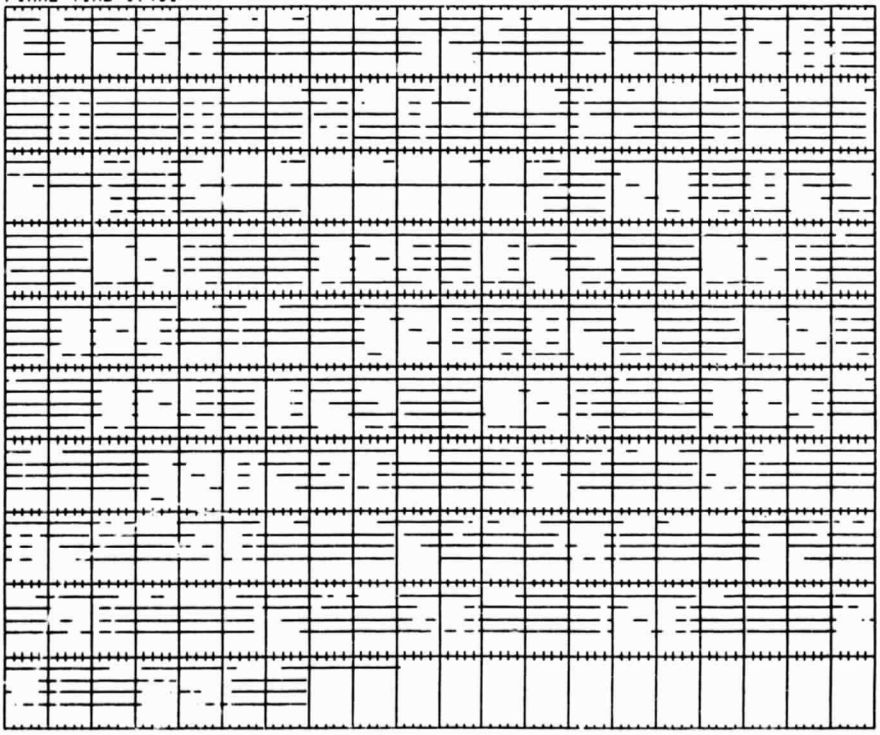


Figure A-2. LSS timeline.

APPROVAL

HARDWARE MATH FOR THE 6502 MICROPROCESSOR

By Ralph Kissel and James Currie

The information in this report has been reviewed for technical content. Review of any information concerning Department of Defense or nuclear energy activities or programs has been made by the MSFC Security Classification Officer. This report, in its entirety, has been determined to be unclassified.

W. C. BRADFORD
Director, Information and Electronic
Systems Laboratory

☆ U.S. GOVERNMENT PRINTING OFFICE 1985—544-048/20002