The SY6516 Pseudo-16 Bit Processor

While the 6502 is a great microprocessor as it stands, advances are being considered to make it even better. One of the approaches is to add some new capabilities such as some 16 bit operations, improved addressing, and more.

Randall Hyde
12804 Magnolia
Chino, CA 91710

For those of you who may have wondered what the 6502 equivalent of the MC6809 would be, wonder no longer. Synertek is almost ready to ship the SY6516.

Synertek announced the 6516 almost a year ago, but due to production problems, it never quite made it. The 6516 was designed by Atari Inc. (back then it was to be called the 6509) for use with the Atari 400 and 800 computer systems. Unfortunately, Synertek was unable to deliver the chip in time for Atari to use it in their computers.

What is a Pseudo 16-bit Computer?

A pseudo 16-bit computer uses an internal 16-bit register arrangement, but externally it uses an eight bit bus. Sixteen bit data is multiplexed in, much like the Alpha Micro computer on the S-100 bus. In addition to the new 16-bit instructions, the 6516 maintains all of the 8-bit instructions of the 6502. You may reassemble your source files currently on the 6502 and run them directly on the 6516. All the information that I have recieved says that the 6516 is SOURCE code compatable with the 6502 and that it is OBJECT code incompatable with the 6502. I have heard rumors that Synertek is attempting to make the 6516 object code compatable, but quite honestly, I don't believe there is much chance of it happening.

Unlike the Motorola MC6809, which has a distinct set of 8-bit instructions and a distinct set of 16-bit instructions, the SY6516 contains a special register (the "Q" register) which toggles the system back and forth between 8-bit operation and 16-bit operation. In addition, all registers in the 6516 (A, X, Y, and SP) are now 16-bits wide. The "Q" register contains four bits which may be programmed to put the accumulator in the 16-bit mode, the X-register in the 16-bit mode, the Y-register in the 16-bit mode, and memory in the 16-bit mode (for use with INC, DEC, ASL, ROL, ROR, LSR, etc.). If the accumulator is programmed to be in the 16-bit mode, then LDA will load the accumulator with 16-bits, the low order byte coming from the specified address and the high order byte coming from the specified address plus one. If the accumulator is in the 8-bit mode, then the LDA instruction behaves identically to the LDA on the 6502. The other registers (X, Y, and Memory) behave identically.

It does not take twice as long to perform a 16-bit instruction compared to the equivalent 8-bit instruction, as you might expect. Usually only one additional clock cycle is required. This means that 6516 code will run as much as 3 times faster than 6502 code performing the same operation.

In addition, several instructions have been "speeded up" over the 6502 equivalent. For instance, implied instructions now only require one cycle for complete execution (the 6502 requires 2). Several other instructions have been speeded up as well (see Table One).

Variety of addressing modes is what makes the 6502 as flexible as it is. The 6516 includes many more addressing modes in its instruction set. In particular, indirect addressing (without the indexed by Y or preindexed by X), 16-bit relative addressing (there is now a jump relative, so your code can be relocatable), and direct page addressing.

Direct page addressing is something really special. It is available on the 6502 in a restricted form; on the 6502 it is called zero page addressing. Direct page addressing is different , in that any of the 256 pages in the 6516 address space may be used. The particular page is selected by the 8-bit direct page register "Z". The direct page facility should clear up many problems associated with zero page conflicts occuring in the 6502.

The New Instructions

The 6516 has a total of 114 instructions (compared to the 6502's 56). This gives a total of 255 different opcodes. Some of the new instructions are listed on the next page.

The User Flag

Bit 5 of the P register has been undefined to this point in the 6502. The 6516 utilizes this bit as a user defined flag. Included in the instruction set are instructions to set and clear this flag, as well as branch if set, and branch if clear. This user defined flag will prove to be a great help to users who are writing a boolean function. Up till now, the 6502 programmer had to use the carry or overflow flag. The user defined flag will help allieviate problems associated with the use of the aforementioned flags.

The 6516 instruction set was defined to allow maximum capability with the minimum number of instructions possible. For those of you who would really like to have seen an instruction of the form:

```
JMP (LBL,X)
```

you may simulate this by:

```
LDY LBL,X
YPC
```

The instruction sequence still requires only 3 bytes (assuming LBL is a direct page reference) and the timing is 7 cycles which is only two cycles more than a straight jump indirect. This would execute just as fast as a JMP (LBL,X) instruction were it included directly in the instruction set.

For those of you who would like to have seen the auto-increment and auto-decrement instructions of the MC6809, once again they can be simulated by the 6516. For instance, the sequence LAX, INX simulates a post increment and INX, LAX simulates a pre-increment. These instructions require two bytes (the same as the 6809) and execute in 3 to 4 cycles (depending on whether you are in the eight-bit or 16-bit mode). This speed is comparable to the 6809.

The only advantage of the 6809 over the 6516 is the 6809 multiply instruction. However, a software multiply on the 6516 should execute fast enough so that it won't make that big a difference.

The addition of two stacks in the 6809 is no real advantage since you can simulate 2, 3 or even n stacks with one 16-bit stack pointer. Those of you writing machine interpreters (such as the UCSD Pascal Pcode interpreter) will be able to simulate a stack machine quite easily on the 6516.

In my opinion, Synertek has taken everything wrong with the 6502 and fixed it, in addition to adding several features which I had not even previously considered. The 6516 is easily the most powerful 8-bit processor available (with due respects to the Intel 8088 which I would rate "almost there"). This opinion, incidently, is not just my own. EDN rated the 6516 above all the 8-bit processors and even some 16-bit processors, several months ago. If Synertek does indeed make the 6516 processor object code compatable with the 6502, it will definitely make the 6516 something you shouldn't scoff at. Why? Because oncethis happen, 50,000 APPLE II computers will be upgradeable directly to a 16-bit processor and maintain software compatability with existing software. Likewise, the 70,000 or so PETs will be upgradeable and the OSI, and the KIM, and of course, the SYM, etc. etc.

The only fault I find with the 6516 is the assembly language mnemonics chosen by Synertek. They should have followed the example laid down by Motorola and used mneomics which specify the action, leaving the decision of where the data is coming from to the operand field.

I am currently writing a version of LISA (an interactive 6502 assembler for the APPLE II) for the 6516. I will maintain Synertek's syntax, however I will add several extensions to the syntax and instruction set to allow a much more regular syntax. This should prove to be a little more pleasant to the die-hard computer scientist.

The New Instructions

LDS	M-)S	(LOAD STACK POINTER FROM MEMORY)
LHA	M-)AH	(LOAD HIGH ORDER ACC FROM MEMORY)
LHX	M-)XH	(LOAD HIGH ORDER X-REG FROM MEMORY)
LHY	M-)YH	(LOAD HIGH ORDER Y-REG FROM MEMORY)
LAX	M(X)-)A	(LOAD ACC INDIRECT THROUGH X REG)
LAY	M(Y)-)A	(LOAD ACC INDIRECT THROUGH Y REG)
SAY	A-)M(Y)	(STORE ACC INDIRECT THROUGH Y REG)
ADD	A+M-)A	(ADD W/O CARRY)
SUB	A-M-)A	(SUBTRACT W/O CARRY)
AXA	A+X-)A	(ADD X REG TO ACC)
AYA	A+Y-)A	(ADD Y REG TO ACC)
AAX	A+X-)X	(ADD ACC TO X REG)
AAY	A+Y-)Y	(ADD ACC TO Y REG)
AMX	X+M-)X	(ADD MEMORY TO X REG)
AMY	Y+M-)Y	(ADD MEMORY TO Y REG)
NEC	NEG(A)-)A	(2'S COMPLIMENT ACC)
RLT		(ROTATE LEFT ACC)
RRT		(ROTATE RIGHT ACC)
ASR		(ARITHIMETIC SHIFT RIGHT ACC)
RHL		(ROTATE AH LEFT THROUGH CARRY)
RHR		(ROTATE AH RIGHT THROUGH CARRY)
RXL		(ROTATE X REG LEFT THROUGH CARRY)
RXR		(ROTATE X REG RIGHT THROUGH CARRY)
RYL		(ROTATE Y REG LEFT THROUGH CARRY)
RYR		(ROTATE Y REG RIGHT THROUGH CARRY)
TZA	Z-)AL	(TRANSFER Z TO ACC LOW)
YPC	Y-)PC	(TRANSFER Y REG TO PC)
PCY	PC-)Y	(TRANSFER PC TO Y REG)
XHA	AL(-)AH	(EXCHANGE ACC BYTES)
XHY	YL(-)YH	(EXCHANGE Y REG BYTES)
XHX	XL(-)XH	(EXCHANGE X REG BYTES)
XXY	X(-)Y	(EXÇHANCE X WITH Y REGISTAR)
	Qx(-)Qy	
SEF	1-)F	(SET USER DEFINABLE FLAG)
CLF	0-)F	(CLEAR USER DEFINABLE FLAG)
LDQ	M-)Q	(LOAD Q REGISTAR FROM MEMORY)
SEV	1-)V	(SET OVERFLOW FLAG)
BFS		(BRANCH IF FLAG SET)
BFC		(BRANCH IF FLAG CLEAR)
JNE		(JUMP IF NOT EQUAL TO ZERO 16-BIT RELATIVE)
JEQ		(JUMP IF EQUAL TO ZERO, 16-BIT RELATIVE)
PHD	A-)(S)	(16-BIT ACC PUSH)
PLD	(S)-)A	(16-BIT ACC PULL)
PHX	X-)(S)	(16-BIT X REG PUSH)
PLX	(S)-)X	(16-BIT X REG PULL)
PHY	Y-)(S)	(16-BIT Y REG PUSH)
PLY	(S)-)Y	(16-BIT Y REG PULL)
PHZ	Z-)(S),	(PUSH Z REG ONTO STACK,
	Q-)(S)	PUSH Q REG ONTO STACK)
PLZ	(S)-)Q	(PULL G FROM STACK,
	(S)-)Z	PULL Z FROM STACK)
PHR		(COMBINATION OF PHD, PHX, PHY, AND PHZ)
PLR		(COMBINATION OF PLD, PLX, PLY, AND PLZ)
BR1		(PERFORMS A JSR ($FFF0))
BR2		(PERFORMS A JSR ($FFF2))
BR3		(PERFORMS A JSR ($FFF4))
BR4		(PERFORMS A JSR ($FFF6))
BR5		(PERFORMS A JSR ($FFF8))