the 6404 MicroCore

the '6404' is an ALU, (4b CLA w/ Subtract), it has 4 Control Flags, xBRI (x, break, Restrict/Reverse, and Interrupt), used to control operations and subroutine/interrupt routing. It has 4 Arithmetic Flags, Negative, oVerflow, Carry and Zero, NOCZ, as normally used here. It has 4-16 'registers', some of which may be vectors. 00 is the accumulator. 1111 is the stack. the magnitude comparator (that can be carried) and a control-state machine that can shift use of the device for different tasks, such as signed vs unsigned addition, i/o processing and control synchronization for shared bus environments. its not a 'better' 6502, its not a 'smaller' 6502, its a 4b mpu, based on the 6502, and a few other chips. its a general purpose chip, its got its turing completeness, it has "more than 16 instructions because of inner control registers", still, its pretty close

Flags	-eXpand-	Branch	Restrict	IRQStop	trans		Carry	Zero	Negative	oVerflow
					any register					
LDA		PLA	PLP	CLI	LDX	LDY	CLC			CLV
STA		PHA	PHP	SEI	STX	STY	SEC			
AND			BIT		TSX	TAY				
ORA					TXS	TYA				
EOR					TAX			unused or modified instructions		
ASL		LSR	ROL		TXA					
		ROR							~~SED~~	
ADC			SBC		INC	DEC			~~CLD~~	
CMP					CPX	CPY				
BCC			BCS		DEX	DEY				
BVC			BVS		INX	INY				
BMI			BPL							
BNE			BEQ							
JSR		BRK		JMP						
RTS				RTI	operations to Stack or Flag Register are done by reading or writing to those registers directly in i/o; either bottom of stack or top of zero page maybe, R bit gives inverse branches if set, B bit controls interrupt/subroutine rounting, Bit op by AND with R set. R bit set on stack sets Register. R = SBC					
NOP										

Figure 1. 6502 ASM reduced to 16 operations, all operations available through use of control flags

The 6404 is a ‘core’ and may not use the same pinouts, configuration and microcode programming or state transition diagrams as other implementations. It is intended for linking and multiprocessor use. It is capable of fast bit manipulations and has a programmable state machine for complex or custom tasks. The program counter, instruction rom and stack can be external to the core and allows for both harvard and von neumann architectures. Its use of a (Moore) state machine in the instruction decoder provides pipelining, along with an instruction counter linked to the address fetch router. The 16 instructions are orthogonal to the 16 registers. The stack is a minimum 16 Words, and can be extended using the Stack Carry lines to control other modules increasing the total depth or width of the stack, though only 1 Word at a time, and 16 Word offsets are available, using the Vector Mapping Table, allows the Target of Register 1111 to be mapped elsewhere, at programming or runtime depending on implementation fabrics used. Registers can be Chained to use Carry, simultaneous read/write, or Rotate commands as needed in the Vector Lookup table. The 6404 is cable of fast serial or parallel communication and control, with some tasks only needing a single 4b instruction counter.

Register Map				
Number Hex : bbbb	**Name of Register**	**16b ALU group?**	**i/o assignable?**	
0 : 0000	W, Omega, Accumulator	00	maybe	
1 : 0001	X, index, Accumulator	00	maybe	
2 : 0010	Y, index, Accumulator	00	yes	
3 : 0011	Z, index, Accumulator	00	yes	
4 : 0100		01	yes	
5 : 0101		01	yes	
6 : 0110		01	yes	
7 : 0111		01	yes	
8 : 1000		10	yes	
9 : 1001		10	yes	
A : 1010		10	yes	
B : 1011	.	10	yes	
C : 1100	Control Status Register	11	yes (MP control)	Multiprocessor use
D : 1101	Data Lookup Register	11	yes (MP control)	Used to set vectors
E : 1110	Math Status Register	11	yes (MP control)	
F : 1111	Stack	11	yes (vector stack)	

These registers are kept in vector look up table that the instruction decoder/predecoder use to control how I/ O is triggered or how carry and rotation occurs. 16b cla groups may be hardwired, and the default ‘fast math’ matrix extensions may be similarly linked. Using a control bit (R+C?) columns in these groups may be rotated or shifted instead of rows, and if linked, can use up to 16b carry look ahead and magnitude comparator circuits to perform these operations more efficiently. Column add/sub and booleans may be possible. Register D has a command used with uncommon flags to enter Data Lookup Table editing mode.

As it stands, there are 8 free registers of 4b each undeclared, and the orthogonal nature of the 6404 architecture allows any of the 16 registers to have any of the 16 base operations (and all of their variants), with the additional features only invoked if configured to do so.

Command	**Control Register**				**Math Register**			
Flags:	R Flag Set	I Flag set	B Flag	X Flag	Negative	oVerflow	Carry	Zero
LOAD, STORE	<change reference instead of value?>						May carry to chained register ?	
JUMPSub RETURN	(reverse jump direction? Other use? Extra addr bit?)	Alters behavior of return and jump, to interrupt	Alter behavior of jump/ return					
ADDwC,	Subtract instead		Used to control signed math?		Used on signed math	Used on signed math Use to control carry on shift?	Used here	Used here
AND, OR, XOR,	NAND, NOR, XNOR							Is set on zero out?
COMPARE	Should this compliment?				Set here	Useful here?	Set here	Set here
ASHIFTL	Rotate instead of shift		Column shift?			Maybe used on rotation, helpful MathReg shifts.	May be used on rotation	
BRANCH(CARRY, SIGN, OVER, COMPARE)		Should branch with I held	Modifies branch					
NOP	Sets flags or not or NOP							

The system instruction decoder has a programmable state machine to offer microcode optimizations and preferences. In implementation fabrics that allow it these can be changed in real time.
So STR_VAL, 00=>1111, put the accumulator value on the stack, , and STR_VAL, 00-=> 001, put the accumulator in register , or "X"
XOR self, 0000, NAND self(0000), 1111. … STR SLF, R set, Complement Self
Performing an operation on ‘self’ should be very fast, probable AND set near ALU in A,B for double load, REG0, should be extra fast, because it has a 00 length address syllable, and is acting directly on itself (contents) with a 00 address type syllable.
having register flags means i can have them be N, V, C, Z, for each register. and use the main registers and flags, for chained operations.

Prototyping ICs List

4x4 register 74HC/HCT670, used for stack and the 4 groups of 4 registers used, various registers

15 base instructions used so far.
Opcode+mod+src+dest
Mod word: AARL Address, Address, Reference, Literal
First crumb is the address word length, 0-3 extra address words,
The next bit controls direct or indirect, is this the value location, or does this contain an offset
Last bit is the literal bit and used to set a literal value instead of a register number in the destination register word
Store 0000 0000 puts zero in the accumulator W (Omega) register?
No extra address length, no relative addressing, not a reference, this actual number, 0000
And thats logical?

(NOP can be substituted with AND self?)
so, each of these can be modified by status flags:, using the BRK, and IRQ flags, controls the JUMPSub instruction. RESERVE turns ASL, into ROTate. AND, OR, XOR, are compliments if this is set instead, so NAND, NOR, XNOR is done.
R doesnt affect LOAD or STORE, though Carry might? Id like a fast comparator in there. rather than a ADC based Compare.

Im not sure i need all those branches, maybe a Branch EQ, reversed with R to NEQ, and use "Branch Mask", which takes an operand, and "ANDs" it with the arithmetic register, and branches based on that. 16 direction branches exist in am2901 land. so you have 'less branch operations' except you get a fast 'branch compare' and a 'branch mask' which i think is more versatile. I didnt add MOVE, because thats just a 'load store'.
TRANSFER, would be interesting, because it could be 'any two registers or addresses', each opcode has an operand argument, if it can use anything other than direct or implied, its 00, in the register, and is 4b+0b, 01, 4+4b, 10, 4b+8b, 11 4b+12b, so you have inherent 16b addressing, or operand fetch counts, as a 2b counter is tied to the opcode fetch register. is my consideration for branching correct?
would a simple 'branch if mask is true', which is really, 'branch if the and from these two registers is not zero', so, Branch Not Zero, is maybe the most basic of Branches?

Im trying for, 16 registers, 16 instructions, 4b, highly configurable/mappable registers and state machine, that chains registers internally, and can bit slice to higher counts, 4, 8, 12, 16, 5x is 20b (1 MiB address), 24, 28, 32, 36, 10x is 40 (1 GiB) ... 64b addressing is 16 of these, and each one is (theoretically) capable of addressing 64b itself (16x 4b registers), so, 256b give or take

D Register (Data page)

Register functions as normal (stack offset?) unless specific Control Flags are set and Command mode entered through OPCODE (JMP ADDRESS LITERAL this REGISTER). The all registers are DATAPAGE REGISTERS. This causes a Vector Pull, Sync or similar pin to be set, to prevent corruption here
In this mode data can be pulled from the stack on register 1111, however when exiting, it needs to be set on each page with its correct values, often (0000) or (1111) if possible Care has to be taken here, as only certain registers operate as normal in each mode and use of control flags is required to get data to place on a register, LOAD/STORE from memory to target register is ok, however commands do not operate as normal in this mode and any register pins assigned to outside vectors, signals from other processors etc go unattended until completed here. NMI signal forces Return Stack Interrupt behavior, and sets mode to normal at that vector address, Return From that NMI, restores states to this mode, values intact. Values are updated on STORE.
Value in D register during this mode is mode table.

Register Control Mode

0000 data page 0, normal operation
0001 data page 1, register link table
0010 data page 2, vector lookup table
0011 data page 3, common flag control
0100 data page 4, matrix link mode table
0101 data page 5, CLA, MC and Column Shift on/off
0110 data page 6, Inner and out Buffer Mapping
0111 data page 7, <other controls>

State machine tables

These each address 16 new register entries, for 16 states.
1000 data page 0, state transition look-ahead
1001 data page 1, state flags set
1010 data page 2, register direction tables
1011 data page 3, output tables
1100 data page 4, alternate tables on or off.
1101 data page 5, multi processor tables
1110 data page 6, extended register length tables, values here act as multipliers of many other routines
1111 data page 7, stack extension tables

Use RTS with Literal Address of Register here, ,after setting Stack, and this register.
RTS here clears most flags, and returns Stack to normal mode, all registers operate normally, and all OPCODES return to normal.

Instruction Phrase

4b opcode + control and math flags + opcode modifier (address length of operand) + source + destination registers.

STORE + 10 (xx) + 00 + 01

Stores the value that is in ram at address (word A, word B + 00 (W register) is stored in the X register.

the Instruction set is 16 operations, with a 2-4b modifier, similar to other architectures,

STORE ADDEND to OPRH-OPRL, 2 cycles+
ADD AUGEND to OPRL, this will perform the ADC to OPRL, and any carry bits will be performed onto OPRH, and both will be 'echoed' to hardware on the next write cycle (these are directly mapped to the 16b 64k hardware space allotted by the Vector Look up table, it is possible this can be set so no further Write operation is needed, and this happens just before, during, or take the place of the hardware Write Cycle, meaning care must be taken to avoid overwrite (the inner control state machine hascontrol priority here)

commands are like Load, SrcRegister, , DestRegister/MMio, ModifierCodes
addresses can be up to 16b long, for 64k addressing on the bus, by default, though could be much larger, using multiple cycles and operation, as well as assigning registers to be address registers in the VLT. addressing uses the 4b word size to look ahead, thus a comparator circuit is important, this can be extended using register vectors, this can be extended with larger word sizes, or through additional/custom register assignment. 256b addressing/operations, 16x4x4 is theoretically possible on this chip without modification, 16,4b registers is 64b, times 4word; addressing this would tax the limit of the system itself and require available addressable ram.

If full address word size multiplier is used in state control table, (ignoring working register size of 4b), using external chaining, and connectivity management circuitry, 4b word, times 16 maximum word multiplier, is 64b register, x 16 linkable registers, 1024 addressing, though this is impractical at best and would no longer be a 6404, it would be a 64x64 and beyond scope, however, this "core" can be extended, to use larger word sizes, such as 8, 12, 16, 20 or 32b. If this is done using external hardware, then this number is used by the microsequencer to control 4b read, write and operation on these registers, however, circuitry must be provided to place this data on the data input bus accordingly. This requires programming the microcode engine at the factory or in the special register modes used to control the microcode state machines.

Program Counter

Program Counter is <probably>external, and uses 4b counters, so, 16, 256, 4k, 64k, instructions, and use individual 4b blocks to jump, look ahead and work with the micro-code instruction handler (4 units deep) and state machine, to control program flow. Base design would allow for up to 16x4b counters, though this is impractical. Interior control of system allows for custom 4b addressing of the external PC, including if lower counters are reset or not. This allows JMP to be performed on any of the 4b PCs, and keeping the lower values intact, or resetting them. 16, 256, 4096, or 64k jumps or higher to be made quickly, by setting the 4b in question directly. Vector lookup allows this to set or reset these to Zero, leave them intact, or use an absolute JMP (PC set.)

The Vector Lookup Tables

there is, a table of links, 16 units, and 2 registers wide, plus a set of up to 4 flags per link, to create carry, comparison or similar operations across multiple bits, so you can say

"Reg 01 +=C> Reg 10" and now reg 01+10 is an 8b virtual register at register 01.;

this means that carry, overflow, etc performed on 5, will carry over or affect register 7, as if it were a single register. The 6404 can use up to 16 registers to carry out a 64b operation, though some of this may be hardware dependent. This uses the Carry Look Up Table to its maximum potential, and will require 16 write cycles to place on the Output Register, 4b at a time.

However mappable registers in the Vector Look Up Table, can reduce this by using hardware mapped Registers, to hold the sum of an 8b or larger number on completion through routing. This extends the 6404 from a 4b processor, up to a 64b processor, with variable register sizes, so long as they are multiples of the base ALU, Inner Hold, Buffer and Datapath Registers and signal lines, there is no waste of cycles or great modification of this platform needed. “Super Processor” with maximum register chaining, is highly impractical outside of optical, phased based or quantum computing fabrics, heat dissipation, wiring complexity and propagation make certain configurations impractical, or superfluous, though the design can be extended to these maximums, if speed of the system allows for all propagation delays and signal timings.

For our scope, the 6404 is a 4b computer that can ‘lock registers’, write to them sequentially, and have them work in tandem. In may cases, creating a 20b Register, may cause the address bus to be active 5 times, when each of the registers is written to, or it will be output one 4b register at a time to external buffers or latches. This kind of flexibility is offered by the VLA.

In such using Zero flag, and the CLA, to determine the 'highest value' in a set of registers quickly. so fast comparisons, are a big features here, sorting, picking etc. This is important in Heaps. this 'should' help with lisp and prolog language implementations

, its a little 4b microslice 6800/6502, and is built to be chained together. with a few borrowed features,

2x is 8b, 4x is 16b, and 16x in a 4x4 grid, is a 16x504 I think, a matrix processor than can do fast 16x16 signed addition, or possibly, fast matrix operations in hardware,

4x4 is pretty good for a matrix, and using 16 digits allows for both the address of the 6404, and its contents to give an address for the number in that 4x4 point in the matrix.

So, some of this, 'is just rules based on sign and such', so computation on the matrix is often quite simple so i might have 'chips', cores, and 2 different actual device conflated here. though i think im on to something.

A way to make The stack is 16 units deep, of 4b + the program counter, plus both flag registers<4 words wide = 128b>.???

so, this 4b chip. its got 16 instructions, similar to the mc14500b, however, instead of the compliments, which are used with the R-flag, you get an ALU instead with a 4b CLA, and a Comparator, which is mostly a staggered set of NORs? and it looks for "which bit in the 4b adder is biggests", so it has a sum and carry, it has a carry, a sum, a carry in, a or b in, or nothing, and it 'selects' the biggest one (you want the comparison for bigger numbers, in a logic look ahead for large numbers, like say to select the largest of 4x 16b numbers really fast)

We may already have this, as the Ti ALU could do most of these
SN54LS181, SN54S181 SN74LS181, SN74S181
If not we can add.
Certainly, we have at least 1, 16b CLA, and LAC., if not 4, or even a 5th, for a 64b add, though 16b is probably sufficient here, the capability is worth looking into

like this: https://www.geeksforgeeks.org/magnitude-comparator-in-digital-logic/

basic pipelining, to control the state machine. it has a table, with 16 entries, for "next state" of the microcode machine, so for "empty cycle" flags all reset. PC kicks (+0): gets opcode from Address LLLL on pc, 3333.2222.1111.LLLL, on set address from Reset Vector Pull. PC +1: opcode=>first part of state machine=>preset inner flags/muxes/state. PC +2: gets target from ram if state , else fetch, fsm=>preset i/o PC +3 gets modifier from ram if state, else get operand from state, else fetch, fsm operate or continue

addressing, else write out if done, reset FSM PC +4-7 : gets data from RAM based on 2b Address counter in FSM from modifier, else fetch. FSM, operate or write,

so its not really 'micro code', its that there is a combinational circuit FSM that sets if flags can change, if the I/O is on, etc, 4b at a time. and because there is variable length addressing, then this 2b (4 word) address counter, works with the FSM and PC to synchronize reads and writes. you can control this, using a look up table, of FSM transitions, and Flags set. so, this is to like do signed vs unsigned addition

or if carry bits are used, if something rotates B or shifts, etc

its like 'the stick shift on the transmission' and you have a few programmable gears, maybe

https://www.uio.no/studier/emner/matnat/ifi/IN3160/v21/timeplan/in3160-l8-2-microcode.pdf

The shift functions are held using Restrict/Reserve"Rotate" flag bit on the control register, determining if the operation is a shift or rotate, and it is possible this is done in one direction, being ASL, and using the R bit to RotateShift Left, a number of places the represent a Shift or Rotation to the Right, if possible to reduce instruction count and control bit usage.

so addressing is 2b long: 00 - implied, no data fetched from memory, 01 - 4b fetch 10 - 8b 2 cycle fetch, 11 - 12b 3 cycle fetch and there is 2b long mode flags 00, 01, 10, 11 these are used to control if the address is Direct, or Indirect, if it uses a value it a register, and other information based on the opcode. A JMP might only change 4 bits, on the 3rd register, and move to a whole different 64k section in that 4k 12b space. and this allows addressing based on the PC, or registers in different ways. or it might move 4bits on the LSW, and reset a small loop, and jmp out of it on a condition, such as testing a counter or strobing a trigger, waiting for an operation to be done, etc

so, there is a way, to get full 16b addressing (or more) here though careful use of external hardware, and the goal is "can I give this a hex digit, and do all kinds of operations to that hex digit, and all of the things i can do, with a controllable set of registers, 4 or 16 4b registers, and maybe some are a Vector address, or the top of the Stack, or the flag registers, so, thats 3 registers there, if its 4 registers, thats 4b to set src and dest registers for STRVAL= $5, 0 reg to 2 reg is 00 +10, so, $2, , this is immediate, no fetch, so 00b on address length, and its in in the registers themselves, no address mode, 00b, so this means the opcode is $52, 0, and the zero is dropped by the state interpreter

Pipelining considerations

variable fetch length, we may use a multi channel, multi stage pipeline in and out here,

so it doesnt 'waste cycles', part of the opcode tells the 'input decoder' how long to wait. like, its integrating this or bridging this information, to help optimize the PC=>instruction fetch cycles

make it so the longest pre-fetch lines up with the longest number of cycles on an operation there is room for, and try to make as many operations the same length as possible" (or in rational ratios to each other)

we have at least one extra "cycle", from needing two word operations mostly. only a few things, NOP, actually only use one R/W cycle, the rest have (fetch)opcode +(decode)address +(execute)modifier +(memory) data+(write)destination.

and it can be shorter or longer, because addressing and PC length are variable. really, 16b each I think.

so there may be an Input flag, and Output flag, or at least States, for each little mpu/alu, and some kind of vector mapping table, link table, dictionary, state transition table/option table.

addressing, is 16b, or is dependent on the chip; you could have 64b addressing, if you use all 16 4b registers, though im thinking maybe, its only 4 for now as the default, 20b is 1mb, and complies with the full ISA standard; 24b is 16 mb, and will support a 65x816, and offers additional uses there outright.

vectors for i/o is good, 4b i/o is simple and covers a lot of low bit communications, like, make those mappable to different addresses using the look up table, and use them for SPI, serial bit-banging, control lines etc.

yes, you can take a register, and 'map' that register, to an 4 bit location, and its 'echoed' on r/w. the look up registry, will have its permissions so at minimum , 4 of those
so you have things like a way to JMP, and if the relative address of the JMP is in say, counter 2 or 3, you can use that State machine Tables, or some control flag, to reset, or not reset, the lower counters to zero, its 16 commands, and there are like 16 options for each one, some of which are ignored, and some of which are address length or offset, using a 3 word length+register, gives 16b address, and 4 registers, WXYZ, give 16b addressing, and, 16b CLA, with LA-Magnitude Comparators.
so, its really closer to 32+ instructions, or 25 is, a bunch are hidden with control flags, like Branch Equal, vs Branch Not Equal, you just set the R bit to pick compliments like that

anyway, this is the little 6404 im working on, and 2x of them, is probably going to be pretty close to 'a slightly different 6800/6502'

16x of the 6404, can be used to make a 16x504, (or what ever) and you get a really simple matrix calculator i want to try out

https://en.wikipedia.org/wiki/XOR_swap_algorithm
ok, so my matrix unit uses 'swap shifters' or something

start: r3.r2.r1.r0:
swap 03: r0.r2.r1.r3,
swap 12: r0.r1.r2.r3,
swap 02: r0.r3.r2.r1,
so thats, ~9 or 10 operations to 'rotate 4 registers'

and that might be done, like by writing a special code to the ALU register, or something, using a table entry and FSM configuration, makes the 'rotate' command, on any register in that look up table, to cause this to occur, or such; the column shift, it works between cells of each register. And this should allow for some fast hexadecimal matrix operations. Id have to plot it to see if its faster, than writing to and from memory normally to do the same. "probably as long as the column shift is easy"
so, there might be, 4 registers, like 0-3 is WXYZ, F is stack, math and control flags are up in D and E, the PC, *might* be on 9, A, B, C? I/O might be on a register or two, so, 4-7, could be the Matrix Registers, and that still leaves a few for vectors, i/o mapping, buffers etc. These 4, can be scratch registers. or they can make at least, a 2x2 matrix of hexadecimal numbers that have a lot of swap options. idk, ill draw up a diagram, and im thinking, this should be, a 16504 thing, on the 4x4 grid of these 4b alus

even on 4 4b registers, rotating these quickly in hardware, might be something to consider, even if you need some ram to use it on a 4x4 matrix. 4 of these could do so.

we want to avoid special purpose registers and commands if possible, though a few exist, and to preserve general use, certain Control Flag Codes can be used to Alter Specific Commands done to specific Registers, such as the Stack, or the Control flags themselves.

so another thing to consider these 'scratch/swap' registers for, is 16b CLA, which is a hardware feature. so ADC (option set) WXYZ (as an opcode modifier), 4567 = 16 cla add or subtract, with carry/overflow, negative if set, will have the result present, either on those registers, (not a bad idea, as they swap, and can cue up on the output 4b bus) or to the WXYZ, which might STOR, better, though STOR, is a register command, it takes a register to 'put on the write bus to the address'

Different Implementations have different Packages

16 address lines is good, it makes it compatible with the 6502 stuff. and maybe thats, dependent on if you want to map registers to I/O, because one is probably a 4b output/write buffer, and maybe you can have some others become linked to the address buffer, or data buffer, or both, depending on your arrangement, so the 6404 Core, can be used on many different arrangements of buses, timing, addressing, and stack depth or arrangement

so an example package might be

- 4 data
- 4-16 command, for external PC, based on registers assigned
- 4 control (irq, dma_sync, write_hold?, ???),
- 4 signal (carry in, ??, carry out, lock),
- 4 clocks (clock, phi1, phi2, ??? rtc/crystal)?
- 4 power: vcc, gnd, nc, reference?
- 4-20b address (based on registers used)

and so several of these might be controlled internally or externally to the 6404 'core' mpu, with its 4x4 swap/shifter scratchpad. (edited) ; and we can use that several ways, so basically, you can map a lot of the 16 registers in this, internally or externally, with data direction and permissions stored in a look up/combination selector, depending on design

now really, I should be look ing at this, as 8b, though im not sure i need it for DMA, because if done right, I should be able to 'block transfer' like 16b, or even 32b at a time, by 'chaining' registers, though i think 16 is probably enough for this thing, and 8 is fine for my needs

On ISAC:

so for my ISAc chip. i need this, mostly without the math, or the fancy shifters, as a 4 bit 'core' i can use with 4-8 registers, and move 8b around my 'isa bus' with a 6502/6522 and some chips, lcd, button latches for controls, audio chips etc; so, the 'control system' is important here, to configure this device, to mostly be in one of several states (wait, acknowledge, message, copy, write, sort a few things (IRQ priority) and tell the 'on board' 6522 what to do when the 6502 cpu module isnt looking

6522, only uses 16 registers, and uses 4 bit commands, to control 2x 8bit ports, and a ton of other stuff, except its a port expander, it has no logic or control other than the 6502 cpu its addressed on, so, on a fpga or whatever, Ill put one of these, or 2 of these, on one port of a 6522, drop most of the timers and such, and use this as a 'controller' for the 'half a 6522 and this chip' to run dma on a system bus, as permanent master

so I assign:

```
00-01:  WX
02-03:  YZ Data 7 downto 0
    04:  AEN, ALE, T/C, RES
    05:  DREQ/DACK
    06:  IRQS, MEM R//W
    07:  IO, RDY, Clk, R/W
08-BC:  Address 19 downto 0
   DE: Control, Arithmetic Flags
     F:  Stack
```

and these are chained onto the register table and work together, i might need to use the Z reg or some flags to control the mem r/w lines or multiplex something, i still have the CNVRBIx bits, id leave Z for its usefulness here, maybe use X for xternal, or the I bit for the ALE, i think its for that.

so here, we have the basic 4 bit layout for 16 assignable registers to the ISA/XT bus, and controlling a 6522s port A,

if it has a 12b counter, it can move 4k, a DMA 'node'

and other than signalling, syncing and maybe bank swapping for the cpu, look ahead cpu instruction lookahead preparation, so like, it fetch extended address ranges, or uses dead cycles to dma based on cpu instructions

so the programmable state machine, you set flags and behaviors in the state look up table, it has ordered transitions, its not a long cycle, 16 states, however, they can set or read flags (flag read as input on transition, set on state, i think this is a "Mealy machine" type, though you should be able to create both.), so you declare this table, and its 'configurations', then you execute code, so it is like, a bios, firmware or microcode you can set up. There is a limit of 16 base states.

in my case ill have "wait, copy, cpu master, device master, interrupt sort, request/acknowledge, and other task/general purpose." or something, maybe a 'shift communications' and use a serial line, so thats a different state and alters the ASL/ROT command, or something
so the 6404, has a state machine-microcode engine, making it more versatile.

it is possible,the 16 operations, are actually closer to the full 64 instructions of a 6502, if extra bits are used, though atm, only 4 words per operation are considered, on the address word counter is 2b wide

https://bitsavers.org/components/amd/bitslice/Mick_Bit-Slice_Microprocessor_Design_1980.pdf

https://en.wikipedia.org/wiki/XOR_swap_algorithm

mc14500b is an interesting little turing machine.

alot of assembly is the same, stripped down pdp-11

4004

6502/6800

6404 bitslice mpu it has

16 'mappable' registers, including flags,

wxyz, and general purpose.

4b arithmetic flag, 4b control flags (IRQ, BRK, RESERVE, Xtra),

it uses 16 instructions,

16 instruction loop controller, for an Air Conditioner, REG8 wired to unit relays, and X bit of Control Status Register to high temperature sensor.REG2 is used as a working register, it is possible to omit using state machine coding.:

4W 00 BRANCH MASK CNTRL 1000 (branch if MSB of CNTL set, ie X bit), to 02 – test pin for value, goto 02
2W 01 JMP 00 – or goto start
4W 02 OR REG8 0100, REG2, –set register 2 to same as 8 is currently, also with 3rd bit set
3W 03 STORE, REG2 REG8, –turn on Unit, put that value on these register wired to pins
4W 04 BRANCH MASK CNTRL 0000 (branch is MSB Clear), to 06 –test for pin low
2W 05 JMP 04 – goto loop point
4W 06 XOR REG2 0100, REG2 turn off 3rd bit – logically turn of variable
3W 07 STORE REG2 REG8, set value to register turning off pin – set control value to pin
2W 08 JMP 00 – go to the start

31 Words, <128 bits or storage for this 'program', which can possibly be optimized further with direct register addressing if encoded this way in our final core design spec.