Review, plus some new stuff

cmpt-150-arc, David Gerhard.

I. DISCLAIMER

- Some people have asked that I make these review notes available to the class and so here they are, but be aware: these are MY review notes, and as such do not necessarily contain all the information you need to know to pass the final exam. In many cases these notes are intended to trigger MY memory to tell ME what I need to remind you of, so if there's something I know well, I might have one or two notes down here that I intend to expand on. There is no guarantee that these notes will trigger your memory in the same way that they trigger mine.

II. Numbers Chapter 1, skip section 1-3

- binary digit groupings: bit, byte, nybble, word, LSB, MSB
- number systems: base 10, base 2, base 8, base 16, Powers of 2
- converting between base 2 and base 10 and back
- BCD and ASCII

III. Gates & Logic 2-1 to 2-3, 2-6, 2-7

- combinational vs sequential design
- boolean algebra: AND, OR, NOT
- postulates for boolean algebra: Existence, Closure, Identity, Complement, Commutativity, Distributivity
- Additional rules: absorption, demorgan etc
- Gates: AND, OR, NOT, bubbled inputs
- encoders, decoders, multiplexers, demultiplexers
- Gates: NAND and NOR, conversions between NAND and NOR using demorgan
- Gates: XOR and XNOR, parity and equality functions

IV. Minimization & k-maps 2-4,2-5

- algebraeic simplification
- variables, literals, minterms, maxterms
- Complexity evaluation: circuit size vs circuit speed
- minimization using karnaugh maps. 3,4,5 variable maps, don't cares, XOR pattern for k-map
- sum-of-minterms representation, product-of-maxterms representation

Review, plus some new stuff

V. Binary Arithmetic 1-3,3-1,3-2,3-8,3-9,3-10

A. Adding and subtraction

- Addition using carry, bitslice design:
  - Sum = A+B+Cin; carry-out says we have too much to represent.
- subtraction using borrow, bitslice design:
  - Diff = A-B-borrow-in; borrow out says we don't have enough to subtract
- Implementation: addition. Build a full-adder
- Implementation: subtraction. Problems of representation

B. Negative Numbers

- sign-magnitude
- two's complement: MSB has negative weight.
- *** difference between two's complement as a noun and as a verb:
  - noun: a representational form. We can interpret a number as two's complement.
  - verb: to take the negative of a number using the two's complement representational form. The two's complement of 1101 is 0011.
  - We often take the two's complement to find the absolute value of a negative (binary) number so we can write it in decimal.
  - eg: the value of 1101? well, the two's complement of 1101 is 0011=3, so the value of 1101 is -3. Alternatively, the value of 1101 is -8+4+1 = -3.
- In working with HC11, some of you came across the term "one's complement" in the manual. The one's complement of a binary number is the logical complement of that number - flip the bits without adding 1.
- Arithmetic overflow: $V = ci_{n-1} \oplus co_{n-1}$, and means that the result of an arithmetic operation is too big for the representation to hold accurately (pos+pos=neg or neg+neg = pos)
- Adder/subtractor hw: use XOR as controlled complement for input to "B" of full adder.

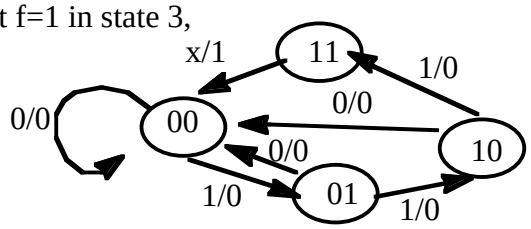
VI. Latches & Flip-Flops 4-1,4-2,4-3

- latch: level triggered. Flip-flop: edge-triggered
- RS latch: cross-coupled NOR / cross-coupled NAND
- D-latch, D-flip-flop, master/slave ff, excitation table / characteristic table
- setup and hold times
- JK flip-flop, excitation table / characteristic table, T-ff (if time)
- asynchronous set and reset

Review, plus some new stuff

VII. Registers and Counters 5-1,5-2,5-3,5-5

- Register, shift register, parallel load shift register
- clock gating versus input multiplexing: spurious edges and clock delay
- Counters, general sequential design:
  - Transition table: for all possible combinations of current state and circuit inputs, show the resulting next state and circuit outputs
  - recall vending machine example.
- *** One more thing here to help us: State Diagrams. These are a convenient way to show the state transitions and inputs/outputs.
- eg: count 0,1,2,3 when c=1, but return to 0 when c=0. output f=1 in state 3, otherwise output f=0.
- From this, draw a transition table, JKffs, k-maps and finally the circuit. (do it)



VIII. TSB, busses, memory (6.1-6.3)

A. TSB

- 3 states: 1, 0, disconnected. Like a switch, used to drive busses.

B. busses

- Collection of wires. Use of multiplexers to drive the bus, or use of TSBs

C. Memory

- RAM: random access (can read any piece of memory in constant time)
- ROM: memory that can't be written to. ROM can be RAM
- Often (and specifically in the HC11) a memory word is a byte in length.
- Memory address: a set of bits that uniquely identify a word of memory
- Address decoder, asserts a single word line to access a single word.
- ROM implementation, RAM implementation. 2^n x k (n addr, k wordsize)

IX. SAP-1 (7-1, 7-2, 9-1)

- Instruction cycle: fetch/execute cycle
- sap-1 format: 4 bits opcode, 4 bits address
- hardware construction by adding hardware instruction by instruction.
- add some stuff on expansion of sap-1 (examples)
  - start from 5-instruction hardware and add "xgba" (B <-> A)

Review, plus some new stuff

- B needs to write to the bus, but we can only do one exchange at a time, so we need a new temporary register to store A while we put B into A, etc etc
- Register transfer notation and control point assignments for each operation in an instruction.
- control sequencer design, invalid instructions
- Assembly language:
  - english mnemonics for machine code
  - pseudo-ops, listing files

X. SAP-2 (8-8, 8-9, 9-4, 9-5, 9-6, 9-8)

- Differences from SAP-1: additional memory, more registers, ALU, bigger addresses, variable operand field size

A. Assembly language organizing principles

- Instruction types
  - Data Transfer: Register/Register and Register/memory
  - Arithmetic / logic: Arithmetic, logical, shift and rotate
  - Flow-of-control: Jump, branch, subroutine
- Operand format, addressing modes

B. SAP-2 instructions

- load/store instructions
- Direct addressing (extended in HC11)
- Immediate operands, Inherent operands, register operands
- Logical inst'ns, shifts and rotates (note difference between SAP-2 rotate and HC11 rotate)
- Flow-of-control instructions: condition codes and flags, Altering the PC.
- Then we did some examples.

XI. HC11 (chapter 6 and appendix A of HC11 manual)

- Block diagram, programmers model
- PC, SP, CCR, A:B=D, X, Y
- 8 condition codes: SXHI NZVC

A. organization of instruction set:

- Data transfer instructions are broken into 3 categories: Loads, Stores and Transfers: (Accumulators A and B, or D); Stack and Index Register instructions; and Condition Code Register Instructions

Review, plus some new stuff

- Arithmetic/Logic instructions are in 5 categories: Arithmetic operations; Multiply and divide; logical operations; Data testing and bit manipulations; and shifts and rotates
- Flow-of-control are in 5 categories: Branches; Jumps; Subroutine Calls and Returns; Interrupt Handling; and Miscellaneous

B. other new concepts in HC11

- Register indirect addressing
- Stack stuff: return addresses, register spilling, temporary storage, parameter passing

XII. Addressing Modes (ch 6 and apx A)

- Terminology: operand, source/destination operand, opcode, addressing mode, effective address (and calculation) Inherent / implied operands vs explicit operands, operand field.
- HC11 is a single address machine.
- Addressing modes: Immediate, Direct/extended, Register, indexed, relative, autoincrement/autodecrement
- Indexed addressing: structured data example
- Branches and jumps using relative and indexed addressing respectively - comparisons.
- long distance conditionals: branch to a jump
- CCR, preserving contents.
- Instructions that don’t change the CCR, dex and inx only affect the “Z” bit, etc.

XIII. Multi-byte and stack. (ch 6 and apx A)

A. multi-byte operations

- extended precision math: adc, sbc
- Shifts and rotates: use of the carry bit.
- multi-byte shifts and rotates: carry bit “connetcs” previous byte to next byte

B. Stack stuff - subroutines

- Stack is a good place to store parameters, local variables - enforces scope
- building a stack frame: push parameters, space for return value, then make the call.
- In the subroutine, store registers then make space for local variables.
- Stack frame: use an index register at the top of the frame, assign suitable offsets.
- Returning from subroutine: undo everything that was done to the stack.
- Quadratic example, Lab example: number of instructions, bytes, cycles.