University of Pennsylvania
Department of Electrical and System Engineering
Digital Audio Basics

ESE150, Spring 2018 — Final — Wednesday, May 2

- Exam ends at 8:00PM; begin as instructed (target 6:00PM)
- Problems weighted as shown at bottom of page.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Provided reference materials on next to last page.
- Show work for partial credit consideration.
- Unless otherwise noted, answers to two significant figures are sufficient.
- Sign Code of Academic Integrity statement (see last page for code).

____

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this exam.

Name: Solution

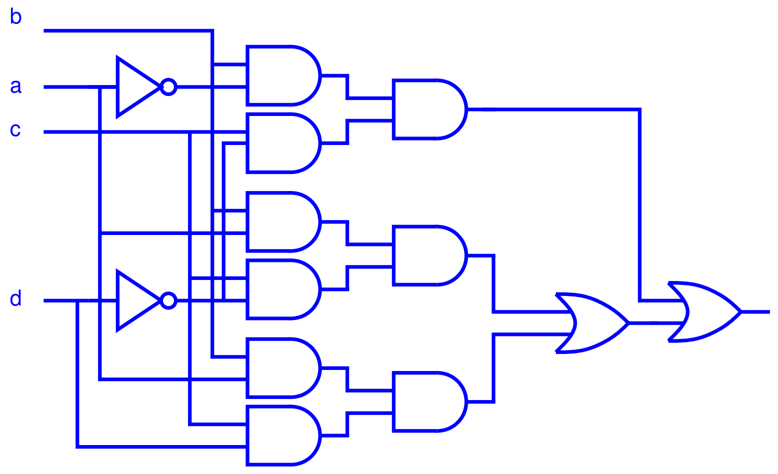
1	2					3						4			
	a	b.i	b.ii	c.i	c.ii	a	b	c	d	e	f	a	b	c.i	c.ii
10	4	2	1	2	1	3	3	3	3	3	3	3	3	3	4

5			6				7				8			**Total**
a	b	c	a	b	c	d	a.i	a.ii	a.iii	b	a	b	c	
2	3	7	3	3	3	3	2	2	2	2	5	5	7	100

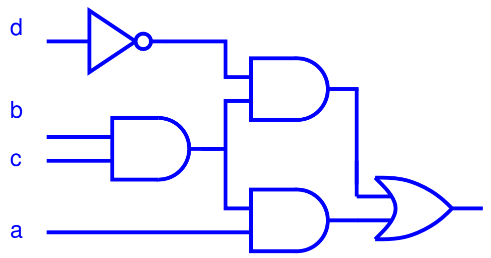
1. Implement the following truth table using inverters and 2-input AND and OR gates.

a	b	c	d	out
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

No Optimization



Optimized



2. We want to move a running computation from one processor to another.

   (a) What information do we need to transfer in order to allow the computation to continue and produce the correct answer?

      i. Program Counter

      ii. Instructions (contents of instruction memory)

      iii. Data (contents of data memory)

   (b) Consider compressing the data before transfer (and decompressing after) using Huffman Encoding.

      i. Will this work (allow the computation to resume on the new processor, run to completion, and produce the same result as if it had not been moved or compressed/uncompressed)?

         Yes. Huffman encoding is lossless, so all state can be restored exactly to the original values.

      ii. How effective is this compression likely to be (Impact on data transfered by component)? Give specific examples where it will be effective or ineffective.

         We will likely to get some compression on instructions due to recurring instruction sequences. Effectiveness on data memory will depend on what's in it. If portions of the memory are 0 (perhaps because they are unused), that will compress well. Any other case where a common value is in memory will also compress well. The program counter is a small part of the state and not necessarily compressible.

   (c) Consider compressing the data before transfer (and decompressing after) using MP3 encoder (decoder).

      i. Will this work (allow the computation to resume on the new processor, run to completion, and produce the same result as if it had not been moved or compressed/uncompressed)?

         No. The MP3 encoding is lossy. If any of the bits in the instruction memory or program counter are changed, the program will not execute as intended. Changes in data state will also lead to incorrect values being computed.

      ii. How effective is this compression likely to be (Impact on data transfered by component)? Give specific examples where it will be effective or ineffective.

         Using fixed-rate budgets for encoding, it can reduce the state; it just won't be restored to useful values when decoded.

3. Consider sending audio data over a UDP (Unreliable Datagram Protocol) link. Assume the receiver is aware of which packets are lost and will make the best of the information it does receive. As in lab, the frequency packets are based on Fourier Transforms of time windows. Rank in order of subjective quality for humans based on your knowledge of audio processing and human psychoacoustics from this course (1-best, 6-worst). Justify your answers by explaining the impact of losing a packet in each case.

   Packets are at most a few thousand bytes – think 2048 sample windows, so cover 10s of ms of sound; a single song is 3 minutes, so will be composed of thousands of packets. Packets are lost independently, even in the case of packet pairs.

   (a) $\boxed{6}$ Packets contain PCM (raw, quantized time-sample data)

   Every lost data packet will result in silence for the length of time covered by the packet.

   (b) $\boxed{3}$ Packets come in pairs for a time window with the highest frequency critical bands in one packet, and the rest in the second.

   In the atypical case in which there are only high or low frequencies, this will be perfect half the time (when the empty frequency set is lost). Typically, there will be both. If the high frequencies are lost, the sound is likely understandable but lower quality; telephone audio only represents data up 4K Hz. If only the high frequency packet is received, this time period will sound odd and any spoken words may be hard or impossible to understand.

   (c) $\boxed{2}$ Packets come in pairs for a time window with the even timeslot PCM samples in one packet of the pair, and odd timeslot PCM samples in the other.

   Losing one packet of a pair will be equivalent to sampling at half the frequency for that time interval. As long as the sound has no frequency content higher than the one-quarter the original sampling rate, the sound will be perfectly reproduced (half for the half lost, half since the Nyquist sampling rate is twice the highest frequency component). If there are higher frequencies present, we will get aliasing effects for the frequencies between one-quarter and half the original sampling rate (assuming the original data was filtered to remove frequencies above half the sampling rate).

   (d) $\boxed{5}$ Packets come in pairs for a time window, were the first packet contains the top 8b of the amplitude for all frequency and the second the low 8b.

It will depend on which packet is lost. If the packet with the low 8b of amplitude is lost, there may be little noise; 8b audio is understandable, and, in many cases, introduces little noise. If the high 8b of amplitude is lost, the time window may sound like noise.

(e) $\boxed{1}$ Packets come in pairs for a time window where both the first and second packet in a pair contains the top one-third frequencies (by amplitude) in each critical band. The first packet also contains the middle one-third of the highest frequency critical bands, while the second packet contains the middle one-third of the remaining critical bands.

Loss is likely unnoticeable. The highest amplitude frequencies in a critical band will mask the lower frequencies. Since the highest (one-third) frequencies exist in both packets in a pair, reconstruction will be able to produce these frequencies even if one of the packets in the pair is lost.

(f) $\boxed{4}$ Packets come in pairs for a time window, where the highest amplitude frequencies from each critical band are in one packet and the lowest in the other.

It will depend on which packet is lost. If the packet with the low amplitude is lost, the result will be unnoticeable, as the high amplitude frequencies will mask the low amplitude frequencies. If the packet with the high amplitude frequencies is lost, the sound will not be reproduced properly. The important components are lost. The remaining components are likely not noise, but they are sounds that would typically not have been heard, so the result is only slightly better than silence.

1 and 6 should not be contraversial. There might be some wiggle room for 2. 3, 4, 5 are more debatable. Pointing out key features of each to support ranking is important thing to get for these.

4. Consider a FLASH memory with the following characteristic:

   - 64KB native flash pages
   - Opening a flash page takes 25μs
   - Once open, access within a flash page is 25 ns per Byte.
   - inodes and bnodes are 1KB blocks
   - must read all bytes in each 1KB block

   (a) How long to read a 38KB file organized as one 1KB inode and 38 1KB bnodes, all of which are placed in the same 64KB flash page?

   $25\mu\text{s}+(25\,\text{ns}\times 1024)\times(38+1)=1{,}023{,}400\,\text{ns}\approx 1.0\,\text{ms}$

   (b) How long to read a 38KB file organized as one 1KB inode and 38 1KB bnodes, each of which is placed in a different 64KB flash page?

   Each of the 39 nodes is in a separate page.

   $(25\mu\text{s}+25\,\text{ns}\times 1024)\times(38+1)=1{,}973{,}400\,\text{ns}\approx 2.0\,\text{ms}$

(c) Consider storing 4 files: A of size 512B, B of size 41KB, C of size 33KB, and D of size 49KB.
(assume all are 1 inode and $\lceil size/1\text{KB} \rceil$ bnodes):

i. How should we place these files into pages in order to minimize the access time for every file? (minimize pages used as a secondary goal.)

Put A and C together in one page, and B and D each in their own pages.

ii. How should we place these files into pages to minimize the number of 64KB pages used? (minimize access time as a secondary goal.) Quantify overall space saving percentage and per file read performance impact percentage.

One page contains all of A and B and 20 1KB bnode blocks from C (a total of 2+42+20=64 blocks); the second page contains all of D and the rest of B (inode block + 13 bnode blocks; so this becomes 50+1+13=64 blocks). This saves 1 of 3 pages or 33% compared to the access-time-optimized version above.

- Access to files A, B, and D is not impacted.
- Access to file C is slowed by 3% ($\frac{920400-895400}{895400} \approx 0.028$)
  - Performance packing: $25\mu\text{s} + 25\,\text{ns} \times 1024 \times 34 = 895{,}400\,\text{ns} \approx 0.90\,\text{ms}$
  - Space packing: $25\mu\text{s} \times 2 + 25\,\text{ns} \times 1024 \times 34 = 920{,}400\,\text{ns} \approx 0.92\,\text{ms}$

5. Consider the following Arduino Assembly code:

```
000010a2 <__udivmodsi4>:
    10a2: a1 e2        ldi r26, 0x21 ; 33
    10a4: 1a 2e        mov r1, r26
    10a6: aa 1b        sub r26, r26
    10a8: bb 1b        sub r27, r27
    10aa: fd 01        movw r30, r26
    10ac: 0d c0        rjmp .+26      ; 0x10c8 <__udivmodsi4_ep>

000010ae <__udivmodsi4_loop>:
    10ae: aa 1f        adc r26, r26
    10b0: bb 1f        adc r27, r27
    10b2: ee 1f        adc r30, r30
    10b4: ff 1f        adc r31, r31
    10b6: a2 17        cp r26, r18
    10b8: b3 07        cpc r27, r19
    10ba: e4 07        cpc r30, r20
    10bc: f5 07        cpc r31, r21
    10be: 20 f0        brcs .+8       ; 0x10c8 <__udivmodsi4_ep>
    10c0: a2 1b        sub r26, r18
    10c2: b3 0b        sbc r27, r19
    10c4: e4 0b        sbc r30, r20
    10c6: f5 0b        sbc r31, r21

000010c8 <__udivmodsi4_ep>:
    10c8: 66 1f        adc r22, r22
    10ca: 77 1f        adc r23, r23
    10cc: 88 1f        adc r24, r24
    10ce: 99 1f        adc r25, r25
    10d0: 1a 94        dec r1
    10d2: 69 f7        brne .-38      ; 0x10ae <__udivmodsi4_loop>
    10d4: 60 95        com r22
    10d6: 70 95        com r23
    10d8: 80 95        com r24
    10da: 90 95        com r25
    10dc: 9b 01        movw r18, r22
    10de: ac 01        movw r20, r24
    10e0: bd 01        movw r22, r26
    10e2: cf 01        movw r24, r30
    10e4: 08 95        ret
```

(a) Identify the top and bottom of the loop.

Loop ends at 0x10d2 where it branches back to the top at 0x10ae. The `brne` is the end of the loop where it conditionally branches back to the top.

(b) How many times does the loop execute?
(Hint: What is the loop exit condition? What register holds the value tested for the exit condition? What happens to this register on each loop iteration? How is the register initialized?)

The full loop executes 32 times, with one part of the loop (0x10c8–0x10d2) executing 33 times.

The loop exits when r1 becomes 0. r1 is holding the value. r1 is decremented (`dec r1`) on every loop iteration. r1 is initialized to 33 in 0x10a2 and 0x10a4, where the `ldi` loads 33 into r26 and moves it to r1.

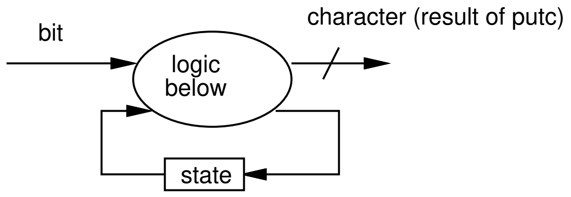
(c) What is the worst-case number of cycles from entering at 0x10a2 to performing the return at 0x10e4?

698

(instruction table on next to last page)

code	cycles	executed	total
10a2: ldi r26, 0x21	1	1	1
10a4: mov r1, r26	1	1	1
10a6: sub r26, r26	1	1	1
10a8: sub r27, r27	1	1	1
10aa: movw r30, r26	1	1	1
10ac: rjmp .+26	2	1	2
10ae: adc r26, r26	1	32	32
10b0: adc r27, r27	1	32	32
10b2: adc r30, r30	1	32	32
10b4: adc r31, r31	1	32	32
10b6: cp r26, r18	1	32	32
10b8: cpc r27, r19	1	32	32
10ba: cpc r30, r20	1	32	32
10bc: cpc r31, r21	1	32	32
10be: brcs .+8	2	32	64
10c0: sub r26, r18	1	32	32
10c2: sbc r27, r19	1	32	32
10c4: sbc r30, r20	1	32	32
10c6: sbc r31, r21	1	32	32
10c8: adc r22, r22	1	33	33
10ca: adc r23, r23	1	33	33
10cc: adc r24, r24	1	33	33
10ce: adc r25, r25	1	33	33
10d0: dec r1	1	33	33
10d2: brne .-38	2	33	66
10d4: com r22	1	1	1
10d6: com r23	1	1	1
10d8: com r24	1	1	1
10da: com r25	1	1	1
10dc: movw r18, r22	1	1	1
10de: movw r20, r24	1	1	1
10e0: movw r22, r26	1	1	1
10e2: movw r24, r30	1	1	1
10e4: ret	4	1	4
			698

6. Consider the following FSM for decoding serial, Huffman encoded data into characters.



```
int state=START;
while(true) {
  int bit=nextInputBit();
  switch(state) {
  case START:  if (bit==0) state=S0; else state=S1; break;
  case S0:  if (bit==0) state=S00; else state=S01; break;
  case S1:  if (bit==0) state=S10; else state=S11; break;
  case S00:  if (bit==0) state=S000; else{ state=START; putc('E'); } break;
  case S01:  if (bit==0) state=S010; else  state=S011; break;
  case S10:  if (bit==0) state=S100; else {state=START; putc(' ');} break;
  case S11:  if (bit==0) state=S110; else state=S111; break;
  case S000: if (bit==0) state=S0000; else {state=START; putc('H');} break;
  case S010: if (bit==0) {state=START; putc('R');} else {state=START; putc('S');} break;
  case S011: if (bit==0) {state=START; putc('N');} else state=S0111; break;
  case S100: if (bit==0) {state=START; putc('I');} else {state=START; putc('O');} break;
  case S110: if (bit==0) {state=START; putc('A');} else state=S1101; break;
  case S111: if (bit==0) {state=START; putc('T');} else state=S1111; break;
  case S0000: if (bit==0) {state=START; putc('C');} else {state=START; putc('U');} break;
  case S0111: if (bit==0) state=S01110; else {state=START; putc('L');} break;
  case S1101: if (bit==0) state=S11010; else {state=START; putc('D');}  break;
  case S1111:  if (bit==0) state=S11110; else  state=S11111; break;
  case S01110: if (bit==0) {state=START; putc('B');} else {state=START; putc('P');}
             break;
  case S11010: if (bit==0) {state=START; putc('G');} else {state=START; putc('W');}
             break;
  case S11110: if (bit==0) {state=START; putc('Y');} else state=S111101; break;
  case S11111: if (bit==0) {state=START; putc('F');} else {state=START; putc('M');}
             break;
  case S111101: if (bit==0) {state=START; putc('V');} else state=S1111011; break;
  case S111011: if (bit==0) state=S1110110; else {state=START; putc('K');}  break;
  case S1110110:if (bit==0) {state=START; putc('X');} else state=S11101101; break;
  case S11101101:if (bit==0) {state=START; putc('Q');} else state=S111011011; break;
  case S111011011:if (bit==0) {state=START; putc('Z');} else {state=START; putc('J');}
                break;
             }
  }
```

(a) What is the minimum number of state bits needed to encode `state` for a hardware implementation? Why?
(Hint: different from the 16b or 32b that might be used to encode an `int` in C.)
5b, because there are 26 states. $\lceil \log_2(26) \rceil = 5$

(b) What is the encoding for the following letters:

letter	encoding
A	1100
B	011100
E	001
Y	111100

(c) What letters can be encoded with each of the following number of bits:

bits to encode	encoding
1	
2	
3	E, space
4	H R S N I O A T
5	C U L D
6	B P G W Y F M
7	V K
8	X

(d) Using the encoding, how many bits does it require to encode:

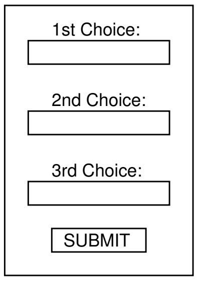
W	E		A	L	L		L	I	V	E		I	N		A	
Y	E	L	L	O	W		S	U	B	M	A	R	I	N	E	

144

7. Consider a smartphone Application User Interface for requesting a partner assignment for lab, where the user gets to submit a prioritized list of 3 choices. Valid selections must be a student in the course which you have not worked with on previous labs.

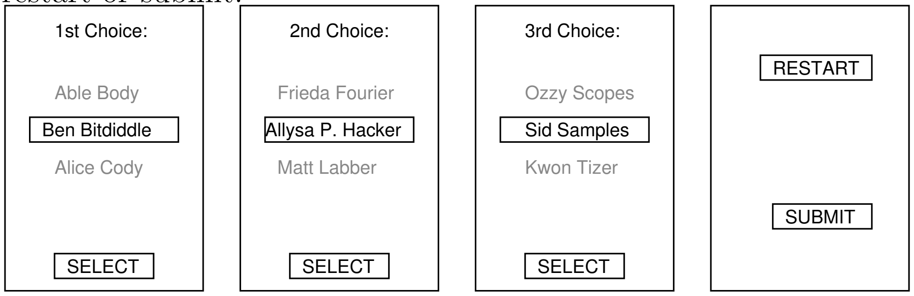
   (a) Considering the UI design issues discussed in class, identify strengths and weaknesses of each of the following (at least one of each):

      i. Text fields to type in 3 names and submit button.



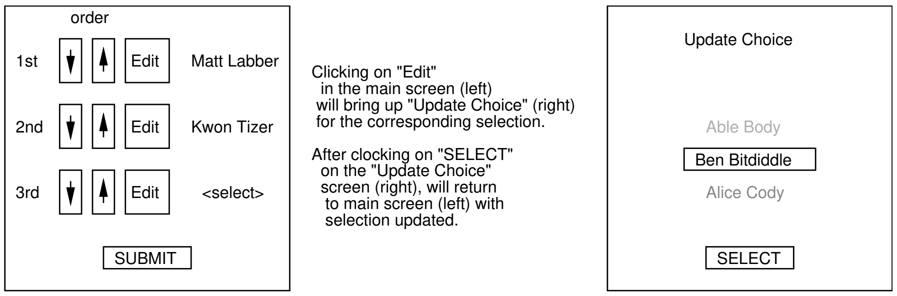
- strength: how to use interface is clear
- strength: can see selection all together before submit
- weakness: does not prevent invalid selections (e.g., students not in class, students who have already partnered with)

      ii. Sequence of three menus/scrolling pickers to select from all students who have not previously partnered (40-n), where n is week of lab. End with option to restart or submit.



- strength: prevents invalid selections—only allows selection from valid list
- weakness: if make mistake or change your mind must start over
- weakness: does not show what selection will be in one place before submit

iii. Default 1st and 2nd selections to highest unassigned student from previous preferences. List current selections in order. Click on selection for scrolling picker to select (update choice). Up/down buttons to order. Submit button.

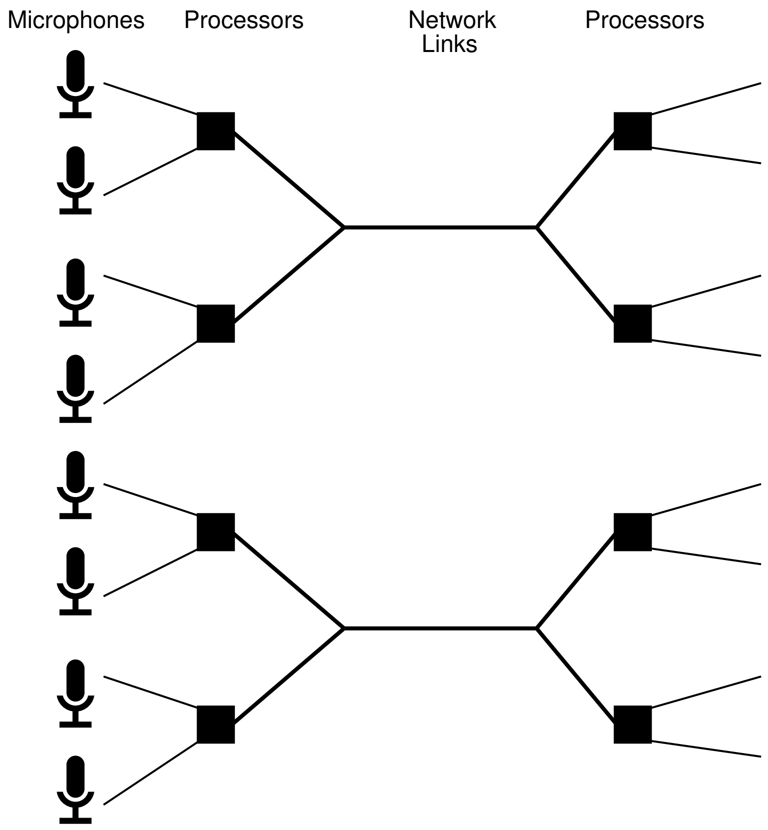

order
1st
Edit
Matt Labber
2nd
Edit
Kwon Tizer
3rd
Edit
<select>
SUBMIT
Clicking on "Edit" in the main screen (left) will bring up "Update Choice" (right) for the corresponding selection.
After clocking on "SELECT" on the "Update Choice" screen (right), will return to main screen (left) with selection updated.
Update Choice
Able Body
Ben Bitdiddle
Alice Cody
SELECT

- strength: prevents invalid selections—only allows selection from valid list
- strength: reduces work by preselecting likely choices; in typical case only need to make one selection rather than 3
- strength: can see selection all together before submit
- weakness: more complicated interface; may be less clear how to use
- weakness: may be too easy to accidentally submit before ready

(b) Suggest an interface that is better than the options above; it may be an improvement to one of the above, a mix-and-match of features, or a completely new design.

- ii – add summary of selections on final selection page; add option to go back and edit a selection
- iii – Default 3rd option to random students, so possibly no need for selection. Add review confirmation screen for submit (maybe it's too easy to accidentally submit?). Show current choices on selection screen. Maybe remove up/down arrows (both or only up or down) to simplify.

8. Consider a scenario where audio data is collected at a number of points and sent over network links to a central location.



- PCM samples: 44KHz sample rate for each input channel at 16b/sample
- For compression (and decompression), each input channel is processed in 2048 point FFT windows
- context switch processor between channels every 2048 samples (matches window size for compression case)
- Processor runs at 1 GHz; can handle many audio channels depending on computation required.
- 20 cycles to handle each uncompressed sample
- 200 cycles of computation per sample for (de)compression
- 20,480 cycles to switch context (channels)
- Frequency transformed and MP3 compressed data is 128Kb/s per channel.
- Network link is 100 Mb/s
- Processor is \$25; network link is \$250.

(a) With one processor on each end and a single network link, how many uncompressed PCM sample channels can the system support? What resource sets the limit? Uncompressed PCM sample channels=150, limited by the bandwidth on the network link.

Processor: $(20+\frac{20480}{2048})$ instructions /sample $\times$ 44,000 samples/sec/channel = 1.3M instructions/sec/channel.

$(10^9$ instructions/sec$)/1.3\times10^6$ instructions/sec/channel $757.57\approx$ 760 channels/processor.

Network link: $(100\times2^{20}\text{b/s})/(44{,}000\times16\text{b/s/channel}) =$ $148.945\approx$ 150 channels/network link.

(b) With one processor on each end and a single network link, how many MP3 compressed channels can the system support? What resource sets the limit? MP3 compressed channels=110, limited by the bandwidth on the processor (de)compression capacity.

Processor: $(200+\frac{20480}{2048})$ instructions /sample $\times$ 44,000 samples/sec/channel = 9.24M instructions/sec/channel.

$(10^9$ instructions/sec$)/9.24\times10^6$ instructions/sec/channel = $108.225\approx$ 110 channels/processor.

Network link: $(100\times2^{20}\text{b/s})/(128\times1024\text{b/s/channel}) = 800$ channels/network link.

(c) What combination of processors, network links, and compression should you use to support 1000 channels at minimum cost? Include the final cost for the solution in your answer.

	Processors each End	Net Links	Cost	
Compressed	1000/110 10	1000/800 2	2×25×10+2×250 \$1,000	Minimum
Uncompressed	1000/760 2	1000/150 7	2×25×2+7×250 \$1,850	

This page intentionally left mostly blank for pagination.
Feel free to use for work space.

Human auditory critical bands:

Band Number	Low	High
1	20	100
2	100	200
3	200	300
4	300	400
5	400	510
6	510	630
7	630	720
8	720	920
9	920	1080
10	1080	1370
11	1270	1480
12	1480	1720
13	1720	2000
14	2000	2320
15	2320	2700
16	2700	3150
17	3150	3700
18	3700	4400
19	4400	5300
20	5300	6400
21	6400	7700
22	7700	9500
23	9500	12000
24	12000	15500

Arduino (AVR) Instructions:

Instruction	Operands	Description	Operation	# Clocks
add	Rd, Rr	add registers no carry	Rd ← Rd+Rr	1
adc	Rd, Rr	add registers with carry	Rd ← Rd+Rr+C	1
brcs	k	~~branch if not equal~~ branch if carry set	if (C=1) then PC ← PC+k+1	2
brne	k	~~branch if carry set~~ branch if not equal	if (Z=0) then PC ← PC+k+1	2
com	Rd	One's Complement	Rd ← 0XFF-Rd	1
cp	Rd, Rr	Compare	Rd-Rr	1
cpc	Rd, Rr	Copmare with carry	Rd-Rr-C	1
dec	Rd	Decrement	Rd ← Rd-1	1
ldi	Rd, k	Load Immediate	RD ← k	1
mov	Rd, Rr	Move Between Registers	Rd ← Rr	1
movw	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	1
ret		Subroutine Return	PC ← STACK	4
rjmp	k	Relative Jump	PC ← PC+k+1	2
sub	Rd, Rr	subtract registers no carry	Rd ← Rd-Rr	1
subc	Rd, Rr	subtract registers with carry	Rd ← Rd-Rr-C	1

Code of Academic Integrity

Since the University is an academic community, its fundamental purpose is the pursuit of knowledge. Essential to the success of this educational mission is a commitment to the principles of academic integrity. Every member of the University community is responsible for upholding the highest standards of honesty at all times. Students, as members of the community, are also responsible for adhering to the principles and spirit of the following Code of Academic Integrity.*

Academic Dishonesty Definitions

Activities that have the effect or intention of interfering with education, pursuit of knowledge, or fair evaluation of a students performance are prohibited. Examples of such activities include but are not limited to the following definitions:

A. Cheating Using or attempting to use unauthorized assistance, material, or study aids in examinations or other academic work or preventing, or attempting to prevent, another from using authorized assistance, material, or study aids. Example: using a cheat sheet in a quiz or exam, altering a graded exam and resubmitting it for a better grade, etc.

B. Plagiarism Using the ideas, data, or language of another without specific or proper acknowledgment. Example: copying another persons paper, article, or computer work and submitting it for an assignment, cloning someone elses ideas without attribution, failing to use quotation marks where appropriate, etc.

C. Fabrication Submitting contrived or altered information in any academic exercise. Example: making up data for an experiment, fudging data, citing nonexistent articles, contriving sources, etc.

D. Multiple Submissions Multiple submissions: submitting, without prior permission, any work submitted to fulfill another academic requirement.

E. Misrepresentation of academic records Misrepresentation of academic records: misrepresenting or tampering with or attempting to tamper with any portion of a students transcripts or academic record, either before or after coming to the University of Pennsylvania. Example: forging a change of grade slip, tampering with computer records, falsifying academic information on ones resume, etc.

F. Facilitating Academic Dishonesty Knowingly helping or attempting to help another violate any provision of the Code. Example: working together on a take-home exam, etc.

G. Unfair Advantage Attempting to gain unauthorized advantage over fellow students in an academic exercise. Example: gaining or providing unauthorized access to examination materials, obstructing or interfering with another students efforts in an academic exercise, lying about a need for an extension for an exam or paper, continuing to write even when time is up during an exam, destroying or keeping library materials for ones own use., etc.

* If a student is unsure whether his action(s) constitute a violation of the Code of Academic Integrity, then it is that students responsibility to consult with the instructor to clarify any ambiguities.