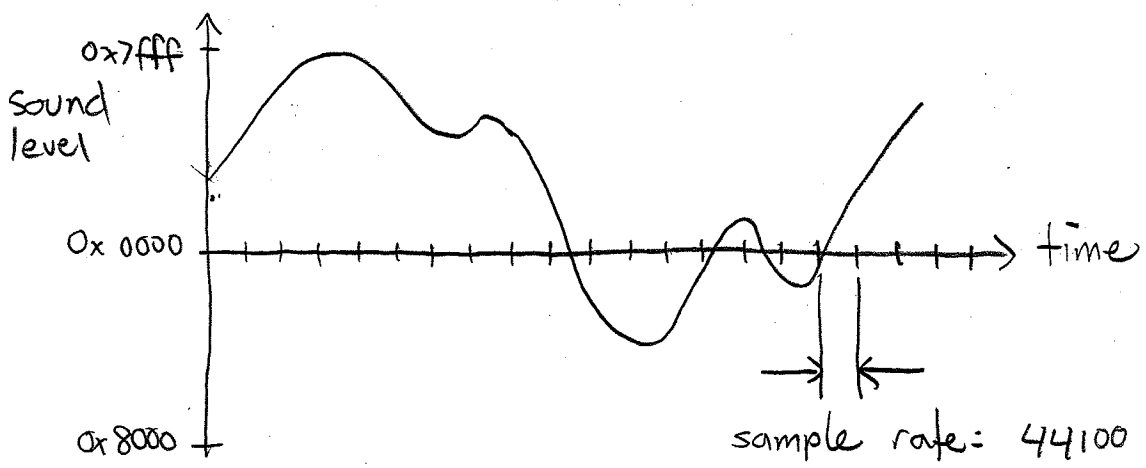
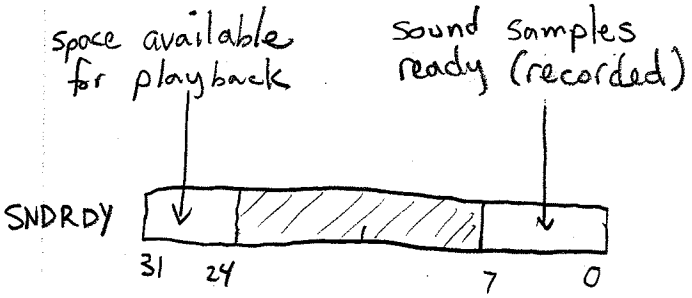
Audio data



sample rate: 44100 Hz

period = $\frac{1}{f} = \frac{1}{44100} \cong 22\mu s$



why 44.1 kHz?

humans hear 20 - 20,000 Hz

need to sample > 2x faster

play sound

```
playsound: ldwio   r2, SNDRDY(r23)
           srli    r2, r2, 24   /* r2 = amount of space available */
           beq     r2, r0, playsound   /* wait for space */
play:      sthio   r4, SNDL(r23)
           sthio   r5, SNDR(r23)
           ret
```

record sound

```
recsound:  ldwio   r2, SNDRDY(r23)
           andi    r2, r2, 0xff   /* r2 = amount of data ready */
           beq     r2, r0, recsound   /* wait for data available */
record:    ldhio   r2, SNDL(r23)
           ldhio   r3, SNDR(r23)
           ret
```

loop to make an "audio wire" from input (microphone) to output (speaker)

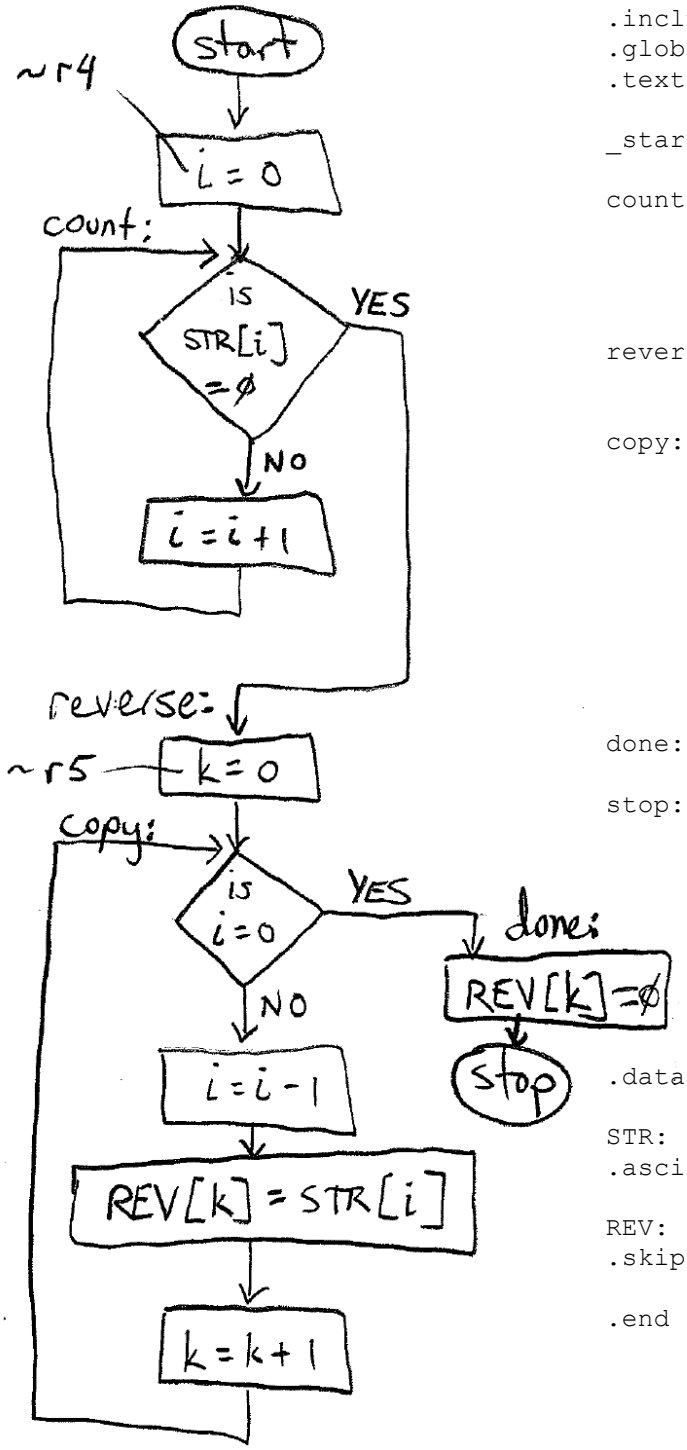
```
/* Play audio data */

.include "ubc-de1media-macros.s"

.global _start

.text
_start:     movia r23, IOBASE

restart:    movia r21, WAV_START    /* play forwards */
            movia r22, WAV_END

nextsound:  ldh   r4, 0(r21)        /* load sound sample from memory */
            ldh   r5, 2(r21)
            call  playsound

            addi  r21, r21, 4             /* go to next audio sample */
            bltu  r21, r22, nextsound     /* address comparisons are unsigned */

            br    restart


playsound:  ldwio r2, SNDRDY(r23)
            srli  r2, r2, 24        /* r2: # samples avail in buffer */
            beq   r2, r0, playsound /* if DAC buffer is full, wait */
            sthio r4, SNDL(r23)     /* send sound sample to DAC */
            sthio r5, SNDR(r23)
            ret


/* Wav data is 16 bit (signed), stereo, sampled at 44.1kHz */

.data

WAV_START:
.include    "wavdata2.s"
WAV_END:

.end
```

```
/* RECORD and PLAYBACK WITH DELAY.
 * record sound from "line in" into a memory buffer.
 * while recording, play back the old contents of the memory buffer.
 * this causes a delay between recording and playback.
 */

.include "ubc-de1media-macros.s"

.global _start

.text
_start:     movia r23, IOBASE

restart:    movia r22, WAV_START
            movia r21, WAV_END

loop:       /* read OLD sounds sample from memory buffer
             */
            ldh   r4, 0(r22)        /* LEFT  */
            ldh   r5, 2(r22)        /* RIGHT */
            call  playsound

            /* write NEW sample to memory buffer
             * this remembers the sound until later,
             * giving the effect of a time delay
             */
            call  recsound
            sth   r2, 0(r22)
            sth   r3, 2(r22)

            /* advance one position in memory buffer. after this,
             * r22 will point to the oldest sample in memory.
             */
            addi  r22, r22, 4
            bltu  r22, r21, loop
            br    restart

/* ****************************************************************** */

recsound:   ldwio r2, SNDRDY(r23)
            andi  r2, r2, 0xff      /* r2: # samples ready in buffer */
            beq   r2, r0, recsound  /* if ADC buffer is empty, wait */
record:     ldhio r2, SNDL(r23)     /* get new sound sample LEFT */
            ldhio r3, SNDR(r23)     /*                       RIGHT */
            ret

/* ****************************************************************** */

playsound:  ldwio r2, SNDRDY(r23)
            srli  r2, r2, 24        /* r2: # samples avail in buffer */
            beq   r2, r0, playsound /* if DAC buffer is full, wait */
play:       sthio r4, SNDL(r23)     /* send sound sample to DAC */
            sthio r5, SNDR(r23)
            ret

.data

WAV_START:
.skip (512*1024)
WAV_END:

.end
```

NAME: _____ STUDENT #: _____

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Feb 25, 2011**

1. Draw a flowchart and write the assembly program to copy a string in reverse order from STR to REV.




```
.include "ubc-de1media-macros.s"
.global _start
.text

_start:   movia r4, STR

count:    ldb   r2, 0(r4)
          beq   r2, r0, reverse
          addi  r4, r4, 1
          br    count

reverse:  movia r5, REV
          movia r6, STR

copy:     beq   r4, r6, done

          subi  r4, r4, 1
          ldb   r2, 0(r4)
          stb   r2, 0(r5)

          addi  r5, r5, 1

          br    copy

done:     stb   r0, 0(r5)

stop:     br    stop


.data

STR:
.asciz  "hello"

REV:
.skip   (REV-STR)

.end
```

NAME: _____ STUDENT #: _____

EECE 259: Introduction to Microcomputers **Lecture Quiz** **Feb 25, 2011**

1. Draw a flowchart and write the assembly program to copy a string in reverse order from STR to REV.

```
.include ”ubc-de1media-macros.s”
.global _start
.text

_start:
```

```
.data
STR:
.asciz  ”hello”
REV:
.skip   (REV-STR)
.end
```

NIOS II Instruction Set Summary

<table>
<tr><td colspan="6">ARITHMETIC</td></tr>
<tr><td>add</td><td>addi</td><td></td><td colspan="2" rowspan="4">add <u>rC</u>, rA, rB</td><td rowspan="4">addi <u>rB</u>, rA, Imm16S

ImmS: signed</td></tr>
<tr><td>sub</td><td>subi</td><td></td></tr>
<tr><td>mul</td><td>muli</td><td></td></tr>
<tr><td>div</td><td></td><td>divu</td></tr>
<tr><td colspan="6">LOGICAL</td></tr>
<tr><td>and</td><td>andi</td><td>andhi</td><td colspan="2" rowspan="4">and <u>rC</u>, rA, rB</td><td rowspan="4">andi <u>rB</u>, rA, Imm16U
andhi <u>rB</u>, rA, Imm16U

ImmU: unsigned</td></tr>
<tr><td>or</td><td>ori</td><td>orhi</td></tr>
<tr><td>nor</td><td></td><td></td></tr>
<tr><td>xor</td><td>xori</td><td>xorhi</td></tr>
<tr><td colspan="6">SHIFT AND ROTATE</td></tr>
<tr><td>sll</td><td>slli</td><td></td><td colspan="2">shift left logical</td><td rowspan="5">sll <u>rC</u>, rA, rB
slli <u>rB</u>, rA, Imm5U
rA: value to shift
rB or Imm5U: amount of shift
(amount is always unsigned)</td></tr>
<tr><td>srl</td><td>srli</td><td></td><td colspan="2">shift right logical</td></tr>
<tr><td>sra</td><td>srai</td><td></td><td colspan="2">shift right arithmetic</td></tr>
<tr><td>rol</td><td>roli</td><td></td><td colspan="2">rotate left</td></tr>
<tr><td>ror</td><td></td><td></td><td colspan="2">rotate right</td></tr>
<tr><td colspan="6">MOVES</td></tr>
<tr><td>mov</td><td></td><td></td><td colspan="2">mov <u>rC</u>, rA</td><td>add <u>rC</u>, rA, r0</td></tr>
<tr><td>movi</td><td></td><td></td><td colspan="2">movi <u>rB</u>, Imm16S</td><td>addi <u>rB</u>, r0, Imm16S</td></tr>
<tr><td>movui</td><td></td><td></td><td colspan="2">movui <u>rB</u>, Imm16U</td><td>ori <u>rB</u>, r0, Imm16U</td></tr>
<tr><td>movia</td><td></td><td></td><td colspan="2">movia <u>rB</u>, LABEL</td><td>orhi rB, r0, high bits of LABEL
ori <u>rB</u>, rB, low bits of LABEL</td></tr>
<tr><td colspan="6">LOAD AND STORE</td></tr>
<tr><td>ldw</td><td>ldwio</td><td></td><td colspan="2">reads data memory</td><td>ldw <u>rB</u>, Imm16S(rA)</td></tr>
<tr><td>stw</td><td>stwio</td><td></td><td colspan="2">writes data memory</td><td>stw rB, <u>Imm16S(rA)</u></td></tr>
<tr><td colspan="6">JUMP AND BRANCHES</td></tr>
<tr><td>callr</td><td>call</td><td></td><td colspan="3">callr rA, call LABEL</td></tr>
<tr><td>jmp</td><td>jmpi</td><td></td><td colspan="3">jmp rA, jmpi LABEL</td></tr>
<tr><td>br</td><td>(always)</td><td></td><td colspan="3">br LABEL</td></tr>
<tr><td>beq</td><td rowspan="7">==
!=
>
>=
<
<=</td><td></td><td colspan="3" rowspan="7">beq rA, rB, LABEL</td></tr>
<tr><td>bne</td><td></td></tr>
<tr><td>bgt</td><td>bgtu</td></tr>
<tr><td>bge</td><td>bgeu</td></tr>
<tr><td>blt</td><td>bltu</td></tr>
<tr><td>ble</td><td>bleu</td></tr>
<tr><td colspan="6">COMPARISONS</td></tr>
<tr><td>cmpeq</td><td>cmpeqi</td><td></td><td></td><td rowspan="6">rA == ____
rA != ____
rA > ____
rA >= ____
rA < ____
rA <= ____
(vs. rB or Imm)</td><td rowspan="6">cmpeq <u>rC</u>, rA, rB
cmpeqi <u>rB</u>, rA, Imm16S
cmpequi <u>rB</u>, rA, Imm16U

sets <u>rC</u>=1 or <u>rB</u>=1 if comparison true, 0 otherwise</td></tr>
<tr><td>cmpne</td><td>cmpnei</td><td></td><td></td></tr>
<tr><td>cmpgt</td><td>cmpgti</td><td>cmpgtu</td><td>cmpgtui</td></tr>
<tr><td>cmpge</td><td>cmpgei</td><td>cmpgeu</td><td>cmpgeui</td></tr>
<tr><td>cmplt</td><td>cmplti</td><td>cmpltu</td><td>cmpltui</td></tr>
<tr><td>cmple</td><td>cmplei</td><td>cmpleu</td><td>cmpleui</td></tr>
</table>